'

Avertissement aux étudiants

Ce syllabus a pour but de vous guider dans l'étude. Il est un fil conducteur de ce qui est dit au cours, mais n'en est pas la reproduction intégrale. Il ne remplace donc nullement le cours. Le cours aborde la matière sous un angle différent et c'est à vous de faire le lien entre les deux. De plus, des parties de cours n'y sont pas reprises, c'est le cas du chapitre sur l'ordonnancement.

Veillez donc à accompagner ce syllabus avec les notes prises au cours. Ayez-en une lecture critique et n'hésitez pas à poser des questions : gardez à l'esprit qu'il peut y subsister des erreurs.

Bonne année scolaire.

Note concernant l'édition 2010-2011

pas de modifications majeures

Je tiens à remercier M. Hazard pour m'avoir mis à disposition une bonne partie de la documentation qui a servi à rédiger ce syllabus. Qu'il trouve ici l'expression de toute ma sympathie.

Je remercie également M. Jaumain qui a pris le temps de relire la version 2006 de ce syllabus et de me faire part de ses judicieuses remarques.

Table des matières

1 Introduction

Ce chapitre a pour but d'isoler le fonctionnement du processeur[1] par rapport aux services d'un système d'exploitation. Nous y décrivons « la boucle du processeur » pour une machine fortement simplifiée sur laquelle nous construirons par la suite. Nous voyons notamment le cycle d'interprétation d'une instruction et l'auto incrémentation du registre IP (Instruction Pointer) qui contient l'adresse de la prochaine instruction à exécuter. Nous voyons ensuite comment les premières instructions sont chargées en mémoire dans la phase d'amorce du système.

Beaucoup d'entre vous utilisent déjà couramment un **ordinateur**, au travers de logiciels d'application tels les traitements de texte (word de Microsoft, write de OpenOffice...), les navigateurs internet (explorer de Microsoft, firefox de Mozilla), et une panoplie d'autres logiciels (chat, écoute de musique, ...).
Probablement moins nombreux sont ceux qui ont déjà interagi avec un système d'exploitation au moyen de son langage de commandes (prompt DOS, fenêtre de l'interpréteur de commandes de Windows2000, shell unix, batch files, scripts, JCL...).
Au cours de vos études vous allez interagir avec l'ordinateur sur ces différents niveaux, vous allez aussi écrire vos propres programmes qui utilisent le système d'exploitation de manière plus ou moins masquée au travers de « morceaux de code système » (appels système) et peut-être même serez-vous un jour amené à programmer des parties d'un système d'exploitation.
Savez vous qu'un système d'exploitation (OS : Operating System) est de loin le logiciel le plus important d'un ordinateur ?

Dans ce syllabus, nous parlons indifféremment de programme ou d'utilisateur qui interagit avec le système d'exploitation. En fait la réalité est que le système d'exploitation n'est autre qu'un ensemble de bouts de code mis à la disposition de programmes et que la seule interaction possible est au travers de programmes, notamment les commandes en ligne ne sont autres que des programmes. Il faudra donc entendre « programme » dans tous les cas.

Certains d'entre vous ont peut-être une petite expérience de programmation, mais vous doutez-vous ou vous êtes vous posé la question du pourquoi un programme qui lit à partir d'un fichier, le fait exactement de la même manière si celui-ci se trouve sur un lecteur de disquette, sur un disque dur ou sur votre clé usb ? Et encore, vous êtes-vous déjà demandé comment l'ordinateur retrouve sur le disque le fichier dont vous avez demandé la lecture en spécifiant son nom « c:\hello » ou « / home / mba / hello » ou « AL7.hello » et à quoi correspond son emplacement physique sur le disque ?

1 Processeur est entendu en termes fonctionnels plus qu'un processeur particulier. Nous nous plaçons ici, dans le cadre d'architectures mono processeur.

Si vous êtes parmi les utilisateurs d'un ordinateur, vous avez sans doute aussi l'habitude de télécharger des documents à partir d' internet et même de continuer tranquillement à rechercher sur internet la page suivante à visualiser pendant que le téléchargement est en cours. Mais comment l'ordinateur s'y retrouve-t-il avec une seule mémoire centrale et un seul processeur pour faire tourner plusieurs applications « en même temps » ?

Avant d'essayer de définir le rôle d'un système d'exploitation et, pour se placer dans le contexte d'utilisation de ce dernier, nous allons ouvrir une parenthèse sur la matière d'autres cours (microprocesseurs) et commencer par définir ce qu'est un ordinateur. Regardons cela de plus près.

1.1 L'ordinateur et ses composants

Un ordinateur se compose de

- une « **mémoire centrale** » (RAM)- contient programmes et données
- une « **unité centrale de traitement** » (CPU) - exécute un programme chargé en mémoire centrale
- **unités périphériques** - permettent l'échange de données avec un utilisateur (écran souris), une mémoire de masse (stockage) , un réseau ...

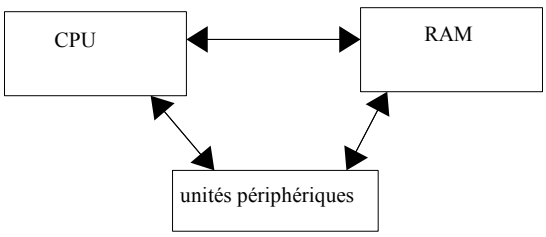


fig 1.1.1.1.1 :les composants d'un ordinateur

1.1.2 Unité centrale de traitement (CPU)

L'unité centrale (processeur) est la partie essentielle d'un ordinateur, elle interprète et exécute les instructions (2) qui se trouvent en mémoire. C'est le cerveau de l'ordinateur(3). Le CPU (Central Proccessing Unit) est constitué essentiellement de

- une Unité de Commande ou de Contrôle (UC) chargée d'extraire de la mémoire l'instruction

2 On parle ici d'instructions de base du processeur. Codées en binaire en mémoire centrale ces instructions de base permettent par exemple de bouger un mot (regroupement de bits) de la mémoire vers un autre emplacement en mémoire.

3 On trouve aussi de nos jours des processeurs spécialisés qui peuvent décharger le processeur et assurer des tâches en parallèle (c'est notamment le cas pour les entrées/sorties). Ces processeurs s'appellent DMA dans le monde des PC.

courante, la décoder et l'exécuter (faisant appel à l'unité de calcul/ALU, si nécessaire).

- une Unité Arithmétique et Logique (ALU) en charge des opérations arithmétiques et logiques

Les principaux éléments de l'Unité de Commande sont :

- le compteur ordinal souvent appelé **IP** (instruction pointer), registre qui contient l'adresse en mémoire de la prochaine instruction à exécuter.
- le registre d'instruction **RI** qui contient l'instruction en cours d'exécution
- le décodeur qui détermine l'opération à effectuer et les opérandes
- le séquenceur qui génère les signaux de commande aux différents composants
- l'horloge (interne ou externe) qui émet des impulsions permettant la synchronisation.

registre :

dispositif de mémorisation temporaire propre à un processeur

L'unité de calcul (ALU) possède, elle aussi, bon nombre de registres dédiés , vous en aurez un aperçu au cours de microprocesseur.

1.1.3 Mémoire centrale (RAM)

Dispositif permettant de conserver pendant un certain temps des informations binaires (données, instructions, ...) : données en provenance de l'extérieur ou destinées à l'extérieur, résultats intermédiaires, instructions de programmes à exécuter. Toutes les informations sont sous forme de **nombres**.

information binaire [4] :

information de type numérique codée en base 2 (utilise uniquement les chiffres 0,1)

lecture du nombre 103 en base 10 : $\mathbf{3}x10^0 + \mathbf{0}x10^1 + \mathbf{1}x10^2$

lecture du nombre 110 en base 2 = $\mathbf{0}x2^0 + \mathbf{1}x2^1 + \mathbf{1}x2^2$ = 6 en base 10 ;-)

Relativement aux registres du processeur nous disposons ici de grandes capacités de stockage mais d'une rapidité d'accès inférieure (+- 10 fois).

La RAM (Random Acces Memory) permet la lecture aussi bien que l'écriture (mémoire vive). Ces mémoires sont dites « adressables à accès aléatoire » c'est à dire que chaque mot peut y être lu ou écrit directement et le temps d'accès à un mot est indépendant de son emplacement, d'où le nom de RAM (Random Access Memory).

4 parfois nous utiliserons le codage héxadécimal : la base 16 qui utilise les 16 chiffres suivants 0 à 9 - A à F.

mot :

unité adressable en RAM. Notion indépendante de la taille (5).

Toutefois, la mémoire vive est volatile : elle nécessite une alimentation électrique pour garder son contenu. Lorsque le courant est coupé, les informations se trouvant en RAM sont définitivement perdues, elle ne peut donc servir au stockage à long terme, pour cela on utilisera plutôt des mémoires auxiliaires basées sur d'autres technologies mais à accès plus lent.

La mémoire centrale est subdivisée en mots (unités adressables) et est reliée à l'unité centrale par deux types de bus :

- le bus d'adresse (véhicule l'adresse du mot mémoire à accéder)
- le bus de données (véhicule le contenu du mot mémoire (instruction ou donnée))

On peut lire et écrire un mot à une certaine adresse de la mémoire; l'écriture efface le contenu précédent du mot.

Tout programme qui s'exécute, se trouve en mémoire, voici le principe de l'exécution d'un programme chargé en mémoire :

1. le programme et les données sont chargés en mémoire
2. IP contient l'adresse de la première instruction du programme.
3. les instructions du programme sont amenées séquentiellement (une par une) dans le registre RI de l'unité de contrôle. Cette dernière analyse l'instruction en cours et déclenche le traitement approprié en envoyant des signaux à l'unité de calcul. Le passage à l'instruction suivante est fait par incrémentation automatique de IP(6).
4. L'IP (Instruction Pointer) contient l'adresse de la prochaine instruction à exécuter. Après chaque utilisation il est automatiquement incrémenté du nombre de mots correspondant à la longueur de l'instruction traitée (7). Le programme est exécuté en séquence. En cas de « rupture de séquence » (branchement, appel d'une routine (module), etc.), l' instruction qui provoque la rupture charge IP avec la nouvelle adresse.

5 Sur intel c'est le byte
6 Le traitement peut nécessiter de faire appel aux unités d'entrée sortie.
7 Il s'agit ici de l'exécution des instructions en séquence

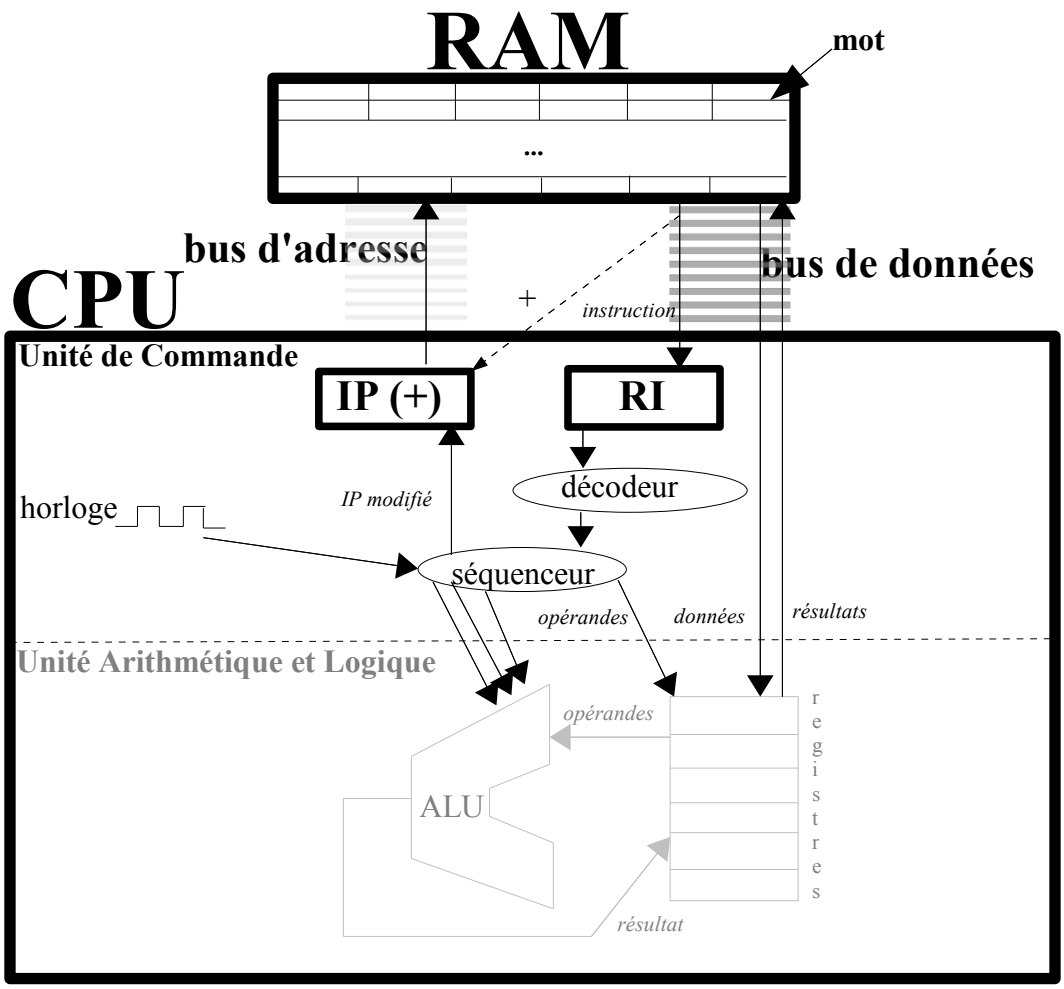


fig 1.1.3.1.1 :schéma simplifié RAM-CPU

Chaque instruction d'un programme est interprétée et exécutée par le processeur de la manière suivante [8]:

- le bus d'adresse reçoit la valeur contenue dans IP (l'adresse de l'instruction à lire)
- RI reçoit via le bus de données, l'instruction dont l'adresse est donnée par IP. IP est incrémenté (adresse de l'instruction suivante)
- l'instruction dans RI est exécutée et on recommence[9] avec la nouvelle valeur de IP.

1.1.4 Périphérique

Dispositif extérieur d'une unité de traitement, permet l'échange avec l'extérieur (imprimantes, clavier, terminaux, modem, ...) ou le stockage d'information (mémoires auxiliaires : disques, bandes, ...).

8 Nous en verrons des exemples au cours.
9 Voilà pourquoi nous appelons ça une « boucle »

1.1.5 En bref

Un ordinateur peut être découpé en blocs fonctionnels, le traitement de l'information se fait au niveau du **processeur**. Les actions que celui-ci doit effectuer sont définies par des **instructions**. Pour être accessibles au processeur, les instructions à traiter et les données doivent être stockées en **mémoire**. Le processeur et la mémoire sont reliés par bus. Par ailleurs, il faut que l'utilisateur puisse échanger avec l'ordinateur des données : instructions à suivre, résultats. Il faut donc des dispositifs d'entrée-sortie (clavier, souris, terminal, disque, bande magnétique...).

Nous interagissons avec un ordinateur au travers de programmes (dont certaines parties sont propres au Système d'exploitation). Pour que un programme puisse s'exécuter, il faut pouvoir le charger en mémoire et que l'emplacement de la première instruction « à traiter » soit chargé dans IP.

Il est à remarquer qu'aujourd'hui nous disposons de programmes système dénommés « **chargeur** » qui nous permettent de charger un programme binaire en mémoire. Le chargeur est donc le premier programme à être chargé en mémoire, cela se fait pendant la « **séquence de boot** » dont nous parlons dans la suite[10].

1.2 Amorce et BIOS

Le processeur exécute les instructions de la RAM, mais au démarrage, celle-ci a un contenu indéterminé. La « séquence de démarrage » ou « amorce » du système a pour but d'amener les premières instructions en RAM et donner à IP l'adresse de la première instruction à exécuter. La suite de ce paragraphe présente cette séquence pour des architectures 80/86).

1.2.1 BIOS et chargeur de démarrage

Lorsque on allume un ordinateur, la borne reset du CPU reçoit un signal logique ce qui provoque une **phase d'initialisation**. IP reçoit l'adresse 1111 1111 1111 1111 1111 1111 1111 0000 (0xfffffff0 en code hexadécimal) qui correspond à une mémoire en lecture seule : anciennement la **ROM** (Read Only Memory), aujourd'hui on utilise des mémoires qui peuvent être réinscrites par logiciel (flashage du BIOS). Les instructions à cette adresse correspondent au BIOS.

Le BIOS Basic Input Output System est un ensemble de programmes qui réside en ROM et qui permet l'accès aux périphériques d'un ordinateur au démarrage d'un système. Le BIOS initie le chargement d'un système, après avoir testé et initialisé les différents périphériques connus, il examine le premier **secteur**[11], secteur 0 ou MBR, de chaque périphérique. Il récupère le code d'amorce du premier périphérique bootable, le charge en mémoire, et lui passe la main. Ce premier secteur d'un disque a donc un rôle particulier, on l'appelle aussi Master Boot Record (MBR). Sur le

10 Nous sommes loin de l'époque où le chargement de programmes était réalisé sur commande d'un opérateur et encore plus de l'époque où la mémoire pouvait être chargée par hardware au moyen d'un panneau d'interrupteurs (un par bit de mémoire pouvant prendre la valeur 0 ou 1). Une mémoire de 256 Mb correspondrait à 2.147.483.648 interrupteurs !

11 Généralement les premiers 512 bytes du disque.

MBR d'un disque dur, 446 bytes sont réservés au chargeur, les 64 suivants contiennent une table à 4 entrées de 16 bytes décrivant chacune une partition du volume (12).

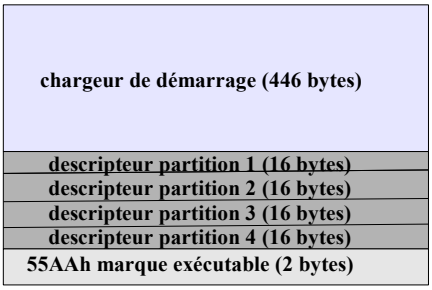


fig 1.2.1.1.1 : secteur 0 ou MBR d'un disque dur

1. le BIOS recherche dans les différents lecteurs du système (CD, floppy, clé USB, Disques durs, ...) le **premier lecteur bootable** (marque exécutable sur le premier secteur du périphérique)
2. le BIOS charge en RAM le premier secteur du premier périphérique bootable. Le premier secteur d'un périphérique bootable contient du code a exécuter, il contient notamment un petit chargeur : le **chargeur de démarrage** (boot loader) (13). Le code du chargeur de démarrage est chargé en RAM à l'adresse fixe : 0000 0000 0000 0000 0111 1100 0000 0000 (0x00007c00 en code hexadécimal), IP reçoit cette adresse, le chargeur de démarrage s'exécute et chargera finalement le système.

Si cette description nous aide à situer notre contexte de travail, elle ne nous aide pas à répondre aux questions que nous avions soulevées en introduction, voyons donc au travers de l'évolution ce qui a justifié la naissance de Systèmes d'exploitation et quels rôles fondamentaux ils ont été rapidement amenés à jouer.

12 Chaque descripteur de partition réserve 32 bits pour stocker la taille de la partition. Ceci limite la taille des partitions MBR à 2 Tb. Pour éviter cette restriction, les BIOS plus récents utilisent une table étendue : GPT (GUID Partition Table) conforme au standard se l'EFI (successeur du BIOS)

13 Lilo, Grub, BootMgr apartiennent à cette catégorie

2 Émergence Et Évolution Des Systèmes D'exploitation

Dans ce chapitre nous évoquons la naissance des systèmes d'exploitation sous plusieurs aspects. Nous voyons en un premier moment la notion de » machine virtuelle » avec les « appels système » ainsi que leur lien avec le mécanisme des « interruptions ». Nous parlons ensuite du « traitement par lots », qui peut être considéré l'ancêtre des langages de script. Nous voyons comment par l'utilisation du « processeur canal », la « multiprogrammation » a vu le jour. Nous abordons enfin la notion de processus et une partie de la problématique du partage imposé par l'exécution de plusieurs processus en mémoire (partage du CPU, partage de ressources, ...)

Les premiers ordinateurs n'avaient pas de Système d'exploitation. Ce n'est que dans le cours des années 50 que les premiers Systèmes d'exploitation ont vu le jour suite à l'évolution de la technologie et au souci de minimiser le coût des ordinateurs (machines très onéreuses à l'époque). Les ordinateurs et les systèmes d'exploitation ont évolué parallèlement depuis ces années, avec eux ont évolué les besoins des utilisateurs.

Notre but n'est pas de retracer l'historique détaillé de cette évolution, nous nous limiterons à mettre en évidence les concepts de base d'un système d'exploitation.

2.1 Machine virtuelle

La première raison d'être d'un Système d'exploitation a certainement été celle de décharger le programmeur de l'écriture d'une série d'instructions très « fastidieuses » nécessaires à la gestion de périphériques (écrire sur un disque ou sur l'imprimante), Ces opérations sont de loin plus complexes de comment les programmeurs les perçoivent aujourd'hui, le système d'exploitation offre au programmeur une « machine virtuelle » : les aspects de plus bas niveau sont masqués (transparence des périphériques). Plus loin, nous verrons des exemples de Système de Fichier : service mis en place par le Système d'exploitation cachant le détail physique du disque en lui-même.

Avant tout voyons comment se présente un disque dur. Un disque est divisé physiquement en pistes et secteurs et des têtes de lecture/écriture mobiles permettent d'accéder l'information des secteurs du disque. Une tête parcourt une piste, chaque opération de lecture et écriture se fait à un emplacement

physique défini par un numéro de piste et de secteur. La lecture nécessite donc un positionnement du bras de lecture préalable. Pourtant, les opérations de lecture et écriture sont, pour les programmeurs, des opérations élémentaires, leurs détails sont masqués (on se limitera à donner l'ordre de lire N bytes (14) dans le fichier nommé « fichier» peu importe où celui-ci réside sur le disque et de quel sorte de disque il s'agit.

Ce sera le rôle du Système d'exploitation d'organiser les données sur le disque et de traduire l'information donnée par le programmeur (un nom de fichier) en adresse physique sur le disque (un numéro de secteur). Les détails physiques du disque sont ainsi masqués au programmeur.

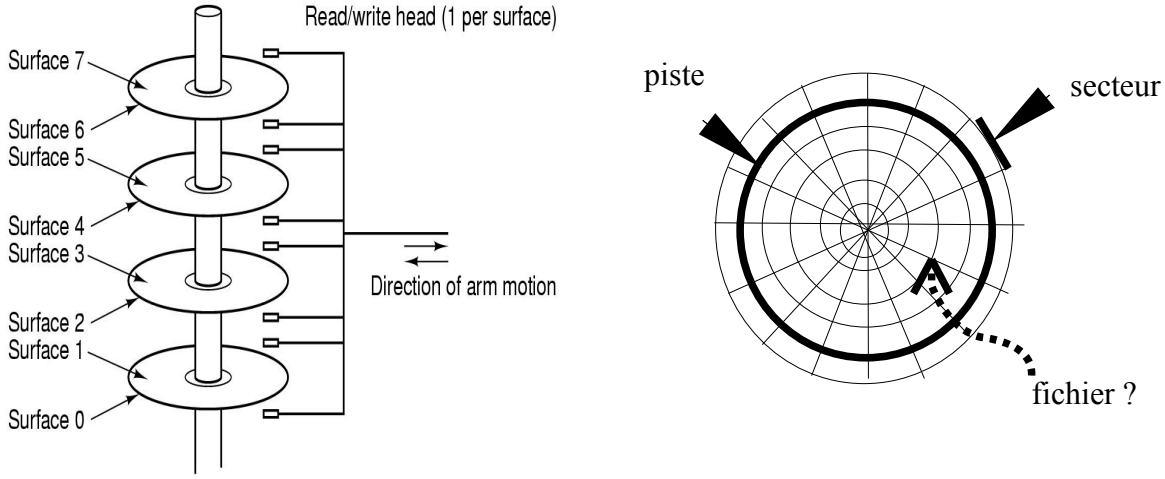


fig 2.1.1.1.1 : constitution d'un disque dur [TNB] et détail d'un plateau

piste :

ligne fictive tracée sur une bande ou un disque magnétique par une tête de lecture-écriture.

secteur :

enregistrement physique constitué d'un ensemble contigu de mots situés sur une piste du disque

14 1 byte = 8 bits

cylindre :

dans une pile de disques, ensemble de pistes qui peuvent être lues sans qu'intervienne un mouvement des têtes de lecture/écriture

L'interaction avec le « Système d' Exploitation » se fait par l'intermédiaire de morceaux de code écrits pour la gestion des « entrées sorties ». Aujourd'hui ces morceaux de code s'appellent « appels système »; ils constituent un ensemble d'instructions étendues fournies par les systèmes d'exploitation. A chacune de ces « instructions étendues » correspondent plusieurs instructions du processeur.

Puisque le système d'exploitation s'occupe de l'accès au matériel et fournit comme interface un « système de fichiers », toute demande de lecture/écriture devra être adressée au Système. Les lectures/écritures de secteurs sur le disque seront des instructions réservées au Système d' Exploitation, c'est facilement compréhensible si on imagine la complexité de l'organisation d'un système de fichiers sur disque.

Apparaît donc la notion d'**instruction privilégiée** : instruction qui ne peut être exécutée que par du « code système » : du code sûr (contrairement au code utilisateur). Avant d'exécuter une telle instruction, le processeur vérifie qu'il est bien en « **mode privilégié**» , un registre du processeur (flag de mode) permet de mémoriser le mode.

2.2 Appels Système et interruptions

Le code d'un appel système ainsi que le code des interruptions s'exécute « **en mode privilégié** » contrairement à un programme « utilisateur » qui s'exécute en « mode utilisateur ». Le mode privilégié donne accès à toutes les instructions du processeur et à toute la mémoire du système. Il existe donc un ensemble d'instructions de base communes plus un ensemble d'**instructions privilégiées**. L'unité de contrôle reconnaît le mode de fonctionnement par un bit d'état qui joue le rôle de « **flag de mode** ».

Seul le processeur et le système d'exploitation ont le droit de modifier ce bit de mode. Dans le cas du traitement d'interruptions le basculement en mode privilégié est automatique (le processeur s'en charge [15]).

Pour passer en mode privilégié, dans le cadre d'une demande de service par un programme (Appel Système), on nécessite une instruction qui puisse être appelée par du code utilisateur en mode « non privilégié ». Par ailleurs, si il existait une instruction non privilégiée ayant comme seul effet de basculer en « mode privilégié » celle-ci permettrait à tout « code utilisateur » de s'exécuter en « mode privilégié » ce qui n'est pas souhaitable.

Les appels système utilisent le mécanisme hardware des interruptions : ce dernier permet un branchement automatique sur du code système (donc du code « sûr ») qui se trouve à une adresse

15 Vu plus en détail au cours

connue (16) par le processeur. Lorsque une interruption est prise en charge, le processeur bascule automatiquement en mode privilégié. Il est important de noter que l'adresse de branchement est déterminée par le processeur et non par le code de l'utilisateur, ce qui permet de garantir que l'on aboutit bien dans du code système.

Une interruption se matérialise par du courant sur un fil. Le processeur vérifie à chaque cycle d'interprétation d'une instruction si une interruption s'est produite. Dans l'affirmative, le processeur bascule en mode privilégié et effectue un branchement à l'adresse où se trouve le code du traitement de l'interruption..

Une instruction particulière permet de déclencher ce mécanisme pour un appel système, sur les architectures 80x86 il s'agit de l'instruction **INT**.

Voici le fonctionnement du processeur intégrant le mécanisme des interruptions :

- **si une interruption s'est produite** sauvegarder IP(17) (il faudra le récupérer plus tard pour revenir au code interrompu), positionner le registre de mode privilégié et mettre dans IP l'adresse du traitement de l'interruption
- le bus d'adresse reçoit la valeur contenue dans IP (l'adresse à lire)
- RI, via le bus de données, reçoit le contenu de la mémoire à l'adresse donnée et IP est incrémenté
- l'instruction dans RI est exécutée et on recommence avec la nouvelle valeur de IP (instruction suivante). Le code qui s'exécute à ce stade est le traitement de l'interruption, dans le cas d'un appel système, ce code effectuera le branchement sur le service demandé.

Les interruptions recouvrent trois mécanismes :

- les déroutements (erreurs d'exécution : division par 0 ...)
- les interruptions externes (évènements externes impromptus : bloc disponible, horloge...)
- les appels système (interruptions logicielles : produites sur demande du programme : demande d' E/S,)

A chaque interruption est associé un morceau de code de traitement qui fait partie du système d'exploitation (18). L'appel système utilise une interruption qui n'est pas générée par des évènements externes, mais par une instruction spécifique sur demande du programme (19).

2.2.1 Retour au code appelant en monoprogrammation

Lorsque une interruption survient, IP prend soudainement(20) une nouvelle valeur qui correspond à l'adresse du code système à exécuter. **Avant cela** le processeur doit se soucier de sauvegarder la valeur de IP pour permettre le retour au code interrompu, le moment venu. Cette valeur est sauvegardée par le processeur, dans une zone de mémoire appelée PILE (STACK) une pile est une

16 Connue ou facilement calculable (sur intel : n° de l'interruption*4)
17 IP est sauvegardé dans une zone de la mémoire (PILE ou STACK)
18 Sur intel ce code est stocké dans un tableau dont les entrées sont indicées par le numéro de l'interruption.
19 Nous verrons en détail au cours le fonctionnement de certains appels système dans le chapitre sur les systèmes de fichiers (demandes de lecture, écriture et ouvertures de fichiers).
20 Le processeur s'en charge. Ceci fait partie du mécanisme hardware des interruptions

zone de mémoire dans laquelle viennent s'empiler successivement des valeurs qui seront réutilisées par la suite dans l'ordre inverse de leur arrivée sur la pile (le dernier entré est le premier sorti), un peu comme une pile d'assiettes(21).

Dans un contexte de monoprogrammation (22) (un seul programme à la fois en mémoire) l'appel système se terminerait en rétablissant le mode « non privilégié » et en restaurant la valeur de IP pour retourner au programme interrompu (23).

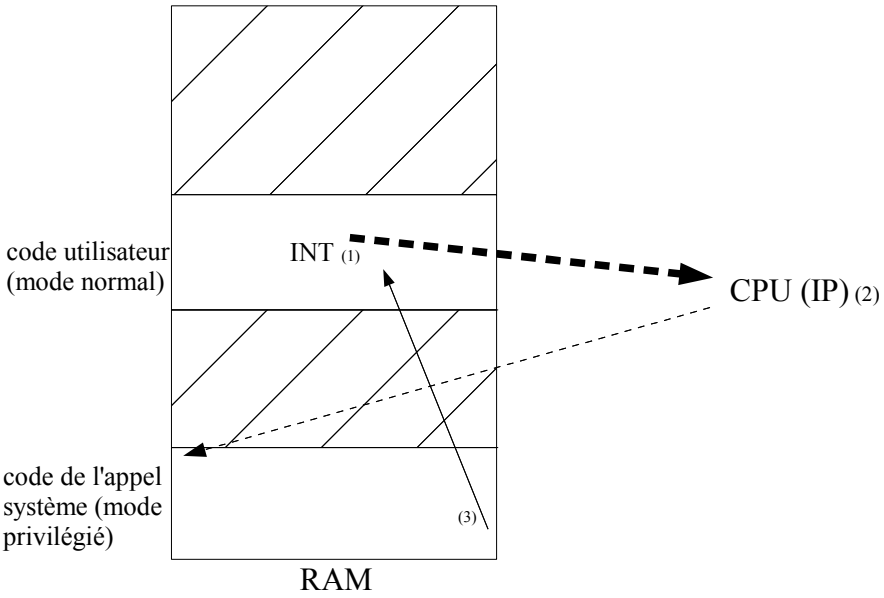


fig 2.2.1.1.1 : un appel système en monoprogrammation :

(1) code utilisateur - l'instruction INT du code utilisateur fait un appel système, il provoque une interruption

(2) CPU - le processeur détecte l'interruption :

1. PILE <-- IP (pointe l'instruction qui suit INT)
2. mode <-- privilégié
3. IP <-- adresse du code de l'appel système

(3) code de l'appel système - la dernière instruction de l'appel système bascule le mode en « non privilégié » et restaure la valeur de IP à partir de la PILE.

1. mode <-- non privilégié
2. IP <-- PILE

21 Pour sauvegarder IP on utilise une zone en RAM pointée par le registre SP (Stack Pointer). Si une interruption survient pendant le traitement d'une autre interruption ou qu'un appel système en appelle un autre (open, read, ...). Nous serons alors confrontés à la nécessité de conserver une cascade de valeurs d'IP pour les retours successifs, voilà pourquoi la zone de mémoire est gérée comme une pile.

22 En multiprogrammation on passe par l'ordonnanceur, c'est lui qui décide à qui rendre la main.

23 Dans le cadre de systèmes en multiprogrammation la main sera plutôt passée à l'ordonnanceur.

2.3 Traitement par lots (batch processing)

En un premier moment, le chargement des programmes en mémoire ainsi que leur enchaînement sera confié à un opérateur humain, l'exécution d'un programme peut prendre plusieurs heures. Nous sommes encore loin des temps de réponse permettant une exploitation interactive de la machine, « on dépose son programme le matin et on revient le soir chercher les résultats, en espérant que tout ait bien marché ». Dans ce contexte la lenteur de l'opérateur humain est relativement insignifiante.

Des Lecteurs et perforateurs de « cartes[24]» sont reliés à l'ordinateur pour permettre les entrées/sorties de données.

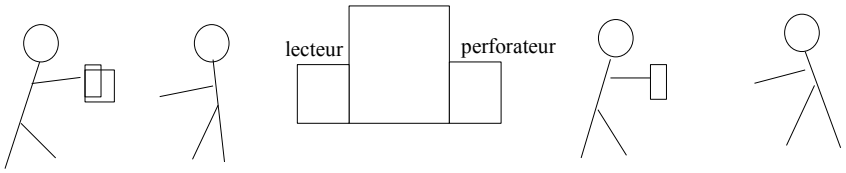


fig 2.3.1.1.1 : un programme sur cartes perforées et résultats quelques heures plus tard

Suite à la diversification et l'augmentation de la vitesse des périphériques, on pourra améliorer le temps de réponse des programmes (temps d'obtention des résultats) et rentabiliser l'utilisation du processeur par la même occasion.

A cette époque, on met au point des logiciels qui permettent d'automatiser une série d'actions manuelles.

Apparaissent les « **moniteurs d'enchaînement** » (élément logiciel sur l'IBM 704) permettant l'enchaînement de différentes opérations au sein de « **travaux** » (jobs). Cette nouvelle technique de travail appelée **traitement par lots** (batch) justifie notamment la mise au point de programmes destinés à placer en mémoire un programme exécutable [25] en vue de son exécution, mieux connus sous le nom de « **chargeur** » (loader).

24 Le paquet de cartes perforées (72 bytes par carte) est le prédécesseur de la clé USB (16 Gb).

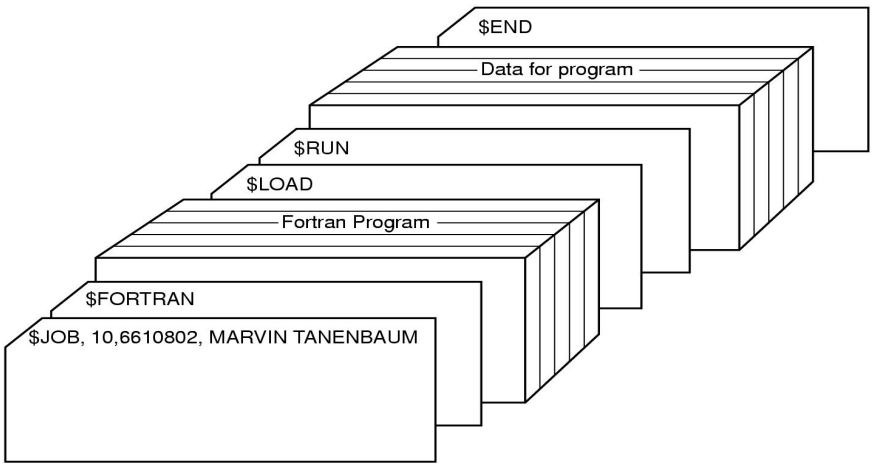
25 Un « programme exécutable » est un programme en code binaire dont on connaît le point d'entrée (adresse de la première instruction à exécuter)

Le traitement par lots est un mode de travail avec enchaînement automatique de commandes sans intervention d'un opérateur. Le programme moniteur doit être capable de réagir en fonction de « cartes de contrôle », en effet tous les travaux ne sont pas identiques et doivent donc être décrits par un programmeur en un « langage de contrôle » connu par le moniteur. De plus ce dernier doit pouvoir enchaîner les différentes étapes en tenant compte d'erreurs qui se seraient produites dans une étape précédente. Les différentes étapes à enchaîner sont décrites dans un fichier de commandes. Le langage comprend une série de commandes dont l'exécution peut être soumise à des codes de condition (si... sinon...), comme un langage de programmation.

Le moniteur assure ici un des rôles essentiels d'un système d'exploitation, dans l'**interprétation des commandes**, leur enchaînement et leur exécution.

Dans la terminologie propre à IBM on parlera de « travail » (job), au sein duquel s'enchaînent des étapes (steps). Un langage dit « langage de contrôle » est utilisé pour décrire un JOB.

Le JCL ((Job Control Language), auquel certains d'entre vous seront confrontés dans le cours de leurs études, en est un représentant, il naît sur l'IBM 360 vers 1964. Y apparaissent différentes commandes (ou cartes car il s'agissait anciennement de cartes perforées): la carte JOB qui permet d'identifier un travail, la carte EXEC qui permet de définir le programme à exécuter ou l'étape du travail, la carte DD qui définit les entrées sorties relatives à une étape.



1. Définition du travail (JOB), notamment de son propriétaire pour pouvoir lui facturer le temps d'utilisation du processeur
2. Exécution de fortran (compilateur) avec en entrée le code source du programme à compiler
3. Chargement en mémoire du programme exécutable produit par l'étape précédente
4. Exécution de ce programme qui utilise des données en entrée

fig 2.3.1.1.2 :structure d'un JOB [TNB]

Nous nous situons ici vers la fin des années 50 à cheval entre la première et la deuxième génération d'ordinateurs. Mais cette notion de traitement par lots existe encore de nos jours, on la retrouve sur OS390, tout simplement parce que ce système d'exploitation est issu de cette génération d'ordinateurs, il est encore utilisé dans de nombreuses entreprises notamment dans des applications

de gestion de salaires (26), facturation, comptabilité mensuelle, enfin dans des traitements de fichiers volumineux, là où le temps de réponse est non critique. Toutefois, l'utilisation de scripts et programmes batch existe également sur des systèmes plus récents : en Windows sous l'appellation de **scripts batch** (fichiers de commandes) ainsi que dans les **scripts Unix** sous la forme de fichiers de commandes shell. Elle y est utilisée notamment pour des tâches d'administration, elle permet par exemple la gestion de tâches automatisées lancées par l'utilitaire « crontab » dans un système Linux(27).

Jusqu'à ce moment les ordinateurs travaillent en un mode d'exploitation caractérisé par le fait que chaque programme s'exécute à son tour dans la mémoire principale, et ce n'est qu'à la fin de l'exécution de l'un que l'on peut charger et exécuter le suivant. Ce mode d'exploitation se nomme « **monoprogrammation** » (28) , la multiprogrammation apparaîtra seulement plus tard.

Un travail correspond donc à l'enchaînement de plusieurs programmes, comme dans l'exemple précédent où se succèdent une compilation et une exécution.

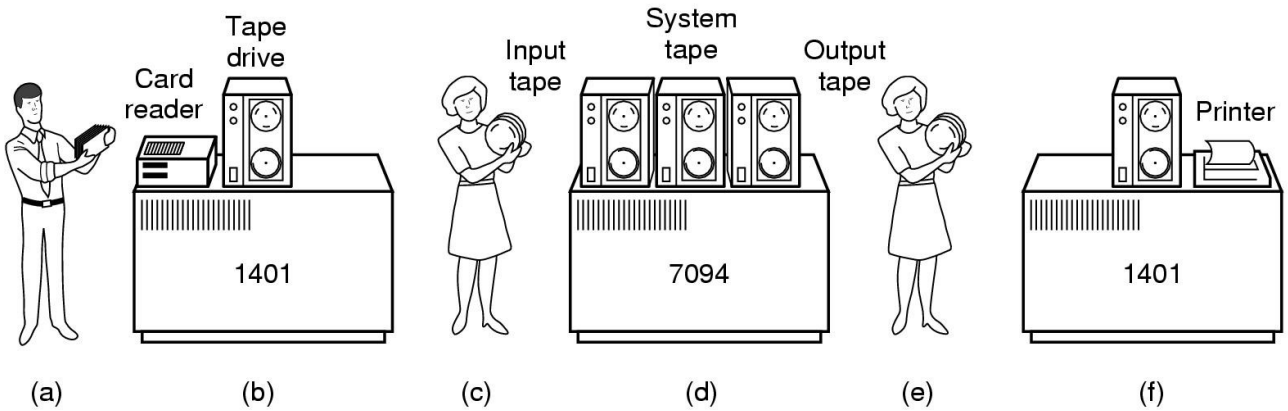
monoprogrammation :

mode d'exploitation d'un ordinateur dans lequel un seul programme est exécuté à la fois dans la mémoire principale. (Dictionnaire Larousse)

Les premiers ordinateurs la plupart du temps, n'avaient comme unités périphériques que des lecteurs et perforateurs de cartes ou bandes perforées, en général, connectés directement à la machine. L'évolution des périphériques et de leur vitesse va amener des changements fondamentaux dans l'architecture des ordinateurs. Au début des années 60 on constatait que l'impression de résultats tout comme la lecture/perforation de cartes ralentissaient l'unité centrale si elles étaient faites à l'aide de périphériques qui lui étaient connectés directement.

Une première amélioration fut d'utiliser des périphériques indépendants, non reliés à l'unité centrale, pour transférer l'information à partir des cartes perforées vers des bandes magnétiques (support plus rapide que les cartes), la transfert inverse étant assuré lors de l'écriture de données (bande magnétique vers carte perforée). La lecture ou l'écriture des bandes aurait été plus rapide et ces opérations de transfert pouvaient se faire de manière parallèle et indépendante de l'UC.

26 Les salaires des enseignants en sont un exemple.
27 Nous illustrerons au cours quelques exemples de job sous OS390 et de script linux.
28 Ce mode d'exploitation se trouve encore aujourd'hui sur les systèmes embarqués qui sont dévolus à une tâche précise et ne s'encombrent pas d'une surcharge de gestion inutile.



a) les programmeurs apportent leurs cartes
b) le 1401 lit l'ensemble de jobs sur une bande
c) l'opérateur apporte la bande sur le mainframe (7094)
d) le 7094 effectue les calculs
e) l'opérateur transfère les résultats sur le 1401

fig 2.3.1.1.3 :ancien système batch [TNB]

bande magnétique :

Support d'information constitué d'une bande de matière plastique souple recouverte d'un oxyde magnétisable. Une bande magnétique est divisée en un certain nombre de pistes (7 ou 9) sur lesquelles sont enregistrées les informations sous forme de suites de bits correspondant à des modifications de l'état magnétique de la bande. (Dictionnaire Larousse)

Une deuxième amélioration était d'utiliser des zones de mémoire (mémoires tampon) (29) accessibles à la fois par le périphérique et l'unité centrale. Un échange d'informations de contrôle entre contrôleur de périphérique et programme devant être assuré en vue de réaliser l'opération de lecture/écriture: emplacement de la mémoire, signaux ou interrogation sur l'état de l'opération (terminaison, problèmes éventuels...)

Dans ce cas, le transfert de l'information depuis et vers la mémoire centrale reste encore assuré par l'UC.

L'ordinateur reçoit du programme en cours l'ordre d'attaquer le périphérique, la lecture sur le périphérique étant lancée il ne reste plus qu'a attendre qu'elle se termine. Le CPU attend la fin de la lecture pour pouvoir continuer l'exécution du programme à l'endroit où on l'a suspendue. Aucune autre instruction n'est exécutée jusqu'à la fin de cette lecture.

t0 t1 t2 t2 + E

29 On retrouve cette utilisation de mémoire tampon aujourd'hui encore dans les imprimantes.

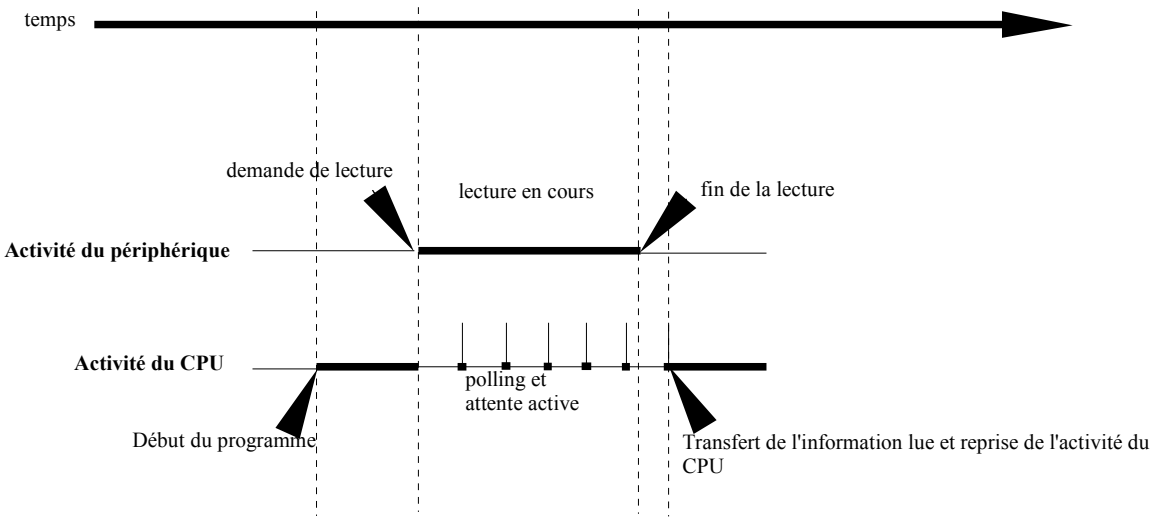


fig 2.3.1.1.4 :schéma de l'activité du périphérique et du CPU pendant une lecture

Cette attente se résume en une attente active, le CPU doit périodiquement interroger le périphérique pour savoir si son travail est terminé c'est ce qu'on nomme « technique de polling » (lecture cyclique d'un état, jusque à ce son état, positionné par le périphérique indique la fin des opérations et la mise à disposition des données).

2.4 Processeur canal

Le programme utilisateur qui occupe le CPU attend pendant la lecture et la vitesse des unités périphériques est très lente comparativement à la vitesse de travail du processeur.

De plus, les ordinateurs coûtent très cher il est donc intéressant de rentabiliser le temps de travail du CPU, des efforts, dans plusieurs directions de travail sont faits pour améliorer ce temps d'utilisation.

Un pas supplémentaire pour améliorer l'utilisation de l'Unité Centrale est franchi lorsque on commence à utiliser les « **canaux simultanés** » (30), dispositifs qui travaillent en parallèle avec l'UC, attaquent directement la mémoire centrale de l'ordinateur en épargnant à l'UC le transfert d'information.

30 Les premiers canaux apparaissent sous forme expérimentale en 1954 sur un IBM 704 pour être commercialisés en 1958 avec l'IBM 709.

canal :

processeur d'entrée-sortie mettant en relation la mémoire d'un système informatique et un ou plusieurs organes périphériques. (Dictionnaire Larousse)

Dans le monde PC on parle aujourd'hui de DMA (Direct Memory Access)

C'est alors que dans l'idée de mieux utiliser l'UC, des essais de simultanéité entre le travail du périphérique et celui de l'unité centrale (lectures anticipées de données) voient le jour.

Dans ce même but, le canal signalera à l'unité centrale la terminaison d'une lecture demandée par **le mécanisme d'interruption**. Ce mécanisme va permettre à l'UC de s'occuper d'autre chose pendant que la lecture suit son cours en parallèle et dès que la lecture sera terminée l'UC en sera avertie. En effet, le mécanisme d'interruption permet d'interrompre le programme en cours pour gérer l'évènement « externe » qui vient de se produire. (exemple : la lecture s'est terminée ou le tampon est plein et il faut le vider pour pouvoir poursuivre la lecture), le polling n'est plus nécessaire.

2.5 Multiprogrammation

Processeur canal et interruptions permettent de décharger entièrement l'UC pendant toute opération d'entrée/sortie, et c'est toujours par souci de rentabiliser l'utilisation des Unités Centrales (éviter les temps d'attente) qu'émerge l'idée d'utiliser l'UC pour un deuxième programme pendant l'attente. Plusieurs programmes pourront coexister en mémoire et être activés tour à tour. Ainsi naît le concept de **multiprogrammation**.

multiprogrammation :

mode d'exploitation d'un ordinateur dans lequel plusieurs programmes résident en mémoire en vue d'une exécution entrelacée.

Dans ce mode de fonctionnement la mémoire de l'ordinateur est partagée (31) entre différents « programmes utilisateurs », plusieurs nouvelles problématiques émergent : comment partager cette mémoire et faire en sorte que chaque programme n'interfère pas avec la zone de mémoire attribuée à un autre. (Il ne s'agit pas d'aller écrire sur les données et encore moins dans le code de quelqu'un d'autre).

En plus d'avoir une quantité de mémoire suffisante pour y charger plus d'un programme, les

31 La manière de partager la mémoire centrale est une des problématiques importantes d'un système d'exploitation , elle fait l'objet d'un chapitre du cours de système de deuxième année, mais vous en aurez un aperçu au laboratoire /cours de Microprocesseurs cette Année.

ordinateurs doivent donc être pourvus d'un mécanisme de protection de l'accès à cette mémoire. Il faut interdire à tout programme d'écrire ou même de lire dans une zone de mémoire qui ne lui est pas attribuée(32).

Dans le schéma suivant nous illustrons l'attribution alternée du CPU à plusieurs programmes faisant des entrées/sorties. L'accès aux canaux est réglé par le « superviseur d'entrée/sortie », quand un programme a besoin d'accéder à un canal il passe par un appel au superviseur. Apparaît ici une nouvelle fonctionnalité du code système : gérer l'alternance des programmes.

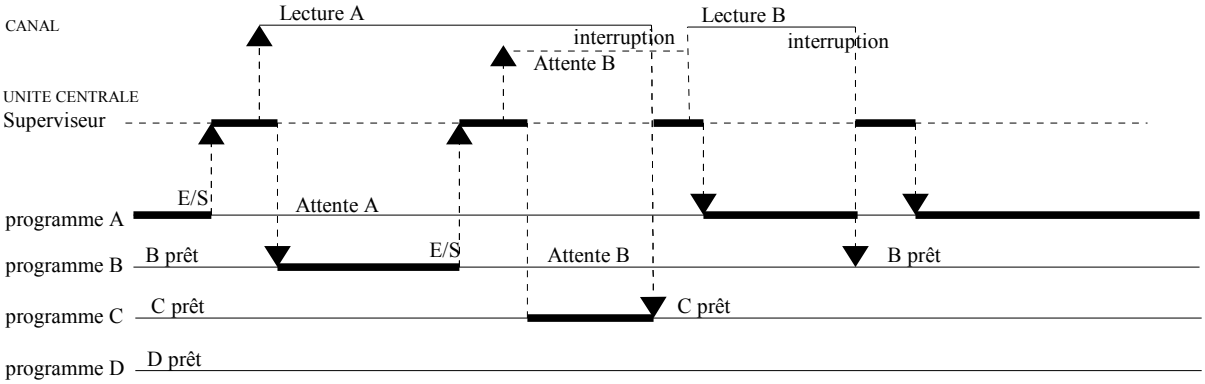


fig 2.5.1.1.1 : partage du CPU et du CANAL entre différents programmes et l'OS (ici nommé superviseur).

Nous constatons dans le schéma précédent que le programme A est avantagé par rapport aux autres programmes, il est traité comme prioritaire. En effet, dès que la lecture qu'il a demandé se termine le CPU lui est nouvellement attribué alors que le programme D n'a toujours pas eu la main même si il est prêt pour s'exécuter. Il en est de même du programme B lorsque son entrée/sortie se termine. Les programmes prêts sont ceux à qui il ne manque plus que le processeur pour pouvoir s'exécuter.

Si nous regardons attentivement cette figure nous constatons qu'un programme qui doit faire beaucoup d'entrées/sorties est pénalisé par rapport à un autre, puisque à chaque entrée/sortie il perd la main et n'aura une chance de la récupérer, une fois l'entrée/sortie finie, que si le programme qui a le CPU fait à son tour une entrée-sortie et si le système le choisit comme prochain programme à activer. Ceci a une répercussion néfaste sur les temps de réponse de ce type de programme

Il est évident que le choix de l'attribution du CPU doit résulter d'une stratégie. Un système conçu pour la multiprogrammation doit disposer d'un **ordonnanceur** : code spécialisé dont un des rôles fondamentaux est de gérer l'alternance des différents programmes qui s'exécutent en mémoire.

La politique du choix du programme prêt auquel attribuer le processeur, est connue sous le nom d'**ordonnancement** (scheduling). Un Ordonnanceur, aura besoin de mémoriser l'état des registres pour chaque processus présent de manière à les restaurer au moment où ceux-ci récupèrent le CPU, ces informations seront mémorisées en mémoire sous la forme d'une table : « **la table des processus** ».

Dans la figure précédente, nous avons introduit le superviseur. Ceci n'est autre qu'une partie du système d'exploitation à savoir la partie de code qui se charge de commander le processeur canal dans le cas d'une demande d' E/S (Entrée/Sortie) ainsi que la partie de code système associée à la fin

32 La segmentation sur intel apporte une solution à ce problème. Voir cours d'assembleur et SYS2.

de la lecture.

Dans un cas nous parlerons d'**appel système**, dans l'autre de traitement de l'**interruption**.

L'ensemble de ces codes système constitue la base d'un Système d'exploitation (33).

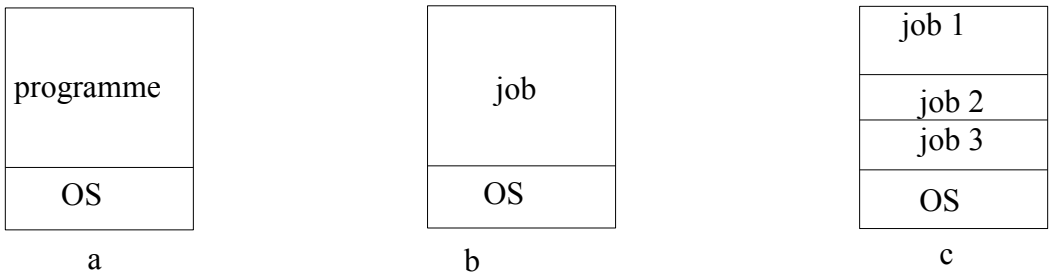


fig 2.5.1.1.2 :occupation de la mémoire centrale dans le cadres de la monoprogrammation a), monoprogrammation en batch b) multiprogrammation en batch c) . OS : dans cette partie de mémoire se trouvent notamment les morceaux de code correspondant aux appels, système, moniteur d'enchaînement et traitement des interruptions.

> *La commande « display » sur OS390 montre les différents jobs de l'utilisateur dans le système et leur état*

2.6 Time slicing et quantum de temps

Tant que tous les programmes sont du même type, ce fonctionnement peut sembler acceptable, mais si par malheur interviennent parmi ceux-ci des programmes faisant pendant la plupart de leur temps des calculs (ne rendent donc jamais la main au système d'exploitation), le temps d'attente des premiers risque d'en être fort affecté. Une autre conséquence est que le Processeur Canal risque d'être la plupart du temps à l'arrêt, ce dernier ne sera donc pas rentabilisé.

Pour éviter ce déséquilibre dans le cas de programmes de natures différentes, on a introduit la notion de « **quantum de temps** » (time-slice). Le time-slice est le temps maximum pendant lequel un programme peut garder le processeur quand on fait du « time slicing ». Une fois ce temps écoulé, le système doit pouvoir récupérer la main pour la donner éventuellement à un autre, un programme faisant travailler le canal par exemple.

Mais comment fait le système pour récupérer la main alors que lui même n'est pas entrain de s'exécuter puisque c'est un programme qui occupe le processeur et que ce dernier n'a aucune intention de rendre la main à qui que ce soit ?

Ce fonctionnement fait appel au mécanisme d'interruption (34), ici l'interruption est provoquée par l'horloge (quantum expiré). La gestion de cette interruption provoque un réordonnancement.

33 Nommé ici Superviseur. En 1964 naît OS (Operating System), système conçu pour l'IBM 360 et avec lui le concept de Système d'Exploitation.

34 Ce mécanisme est vu en détail au cours.

On utilise ainsi un mécanisme de **préemption** pour récupérer la main, on parle de « **multitâche préemptif** ».

2.7 Processus

La notion de processus est un modèle simple pour représenter l'exécution concurrente de tâches au sein d'un système d'exploitation. Elle a été utilisée la première fois avec le système Multics et popularisée depuis.

Un processus modélise l'exécution d'un programme et possède

- un compteur ordinal
- un jeu de registres
- sa zone mémoire (qui peut être virtuelle)
- une information d'état

Un processus alterne trois états :

- prêt : suspendu en faveur d'un autre, il ne lui manque que le CPU
- élu : en cours d'exécution (a le CPU)
- bloqué : en attente d'un événement externe (lecture disque, ressource...)

Le diagramme suivant reprend les transitions possibles d'un état à l'autre

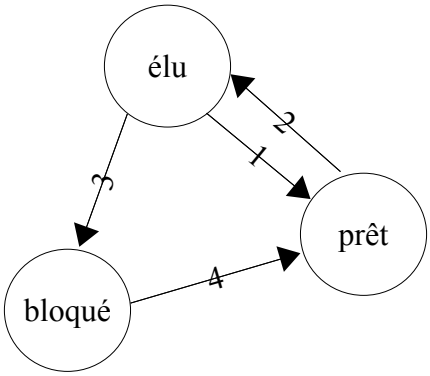


fig 2.7.1.1.1 :diagramme de transition d'états d'un processus

1. Le processus a épuisé le quantum de temps qui lui était attribué. L'ordonnanceur appelé par une interruption horloge élit un autre processus.
2. L'ordonnanceur élit ce processus parmi les prêts.
3. Le processus s'endort en attente d'un événement externe (données, sémaphore, ...).
4. L'évènement attendu par le processus se produit, il est géré par le Système d'exploitation en interrompant temporairement le déroulement du processus élu pour faire passer le processus bloqué à prêt.

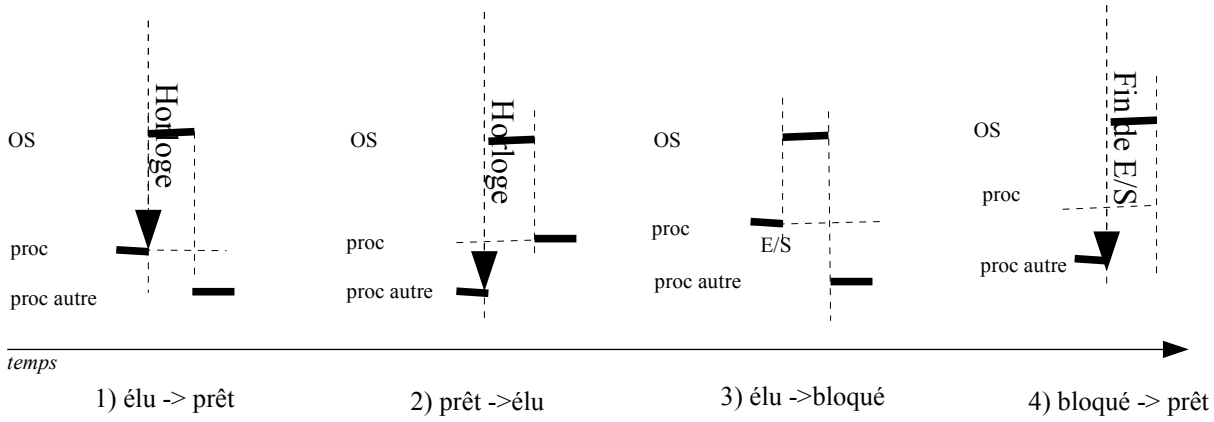


fig 2.7.1.1.2 : vue de l' occupation du CPU pendant les 4 transitions d'état du processus proc

2.8 Accès direct et partage du système informatique

Suite à l'amélioration de la maniabilité et des temps de réponse des ordinateurs, se développe, pendant les années soixante, l'utilisation collective partagée de la machine. Plusieurs utilisateurs : programmeurs, utilisateurs de programmes de traitement de l'information (logiciels) travaillent au même moment sur le même ordinateur. Ce fut l'origine de ce qu'on appelle le « **temps partagé** » (time sharing) technique qui arrivera à maturation au début des années 70. Elle est liée au travail intéractif et a comme corollaire le développement de terminaux permettant d'accéder l'ordinateur à distance.

> **temps partagé** :
>
> mode d'exploitation d'un ordinateur dans lequel chaque utilisateur se voit allouer une tranche de temps fixe et relativement faible (time slice) pour l'exécution de son programme. (A l'issue de cette tranche de temps, si l'exécution n'est pas terminée, le programme est mis en attente, tandis que les autres programmes demandeurs sont exécutés tour à tour. Cycliquement, chaque programme est alors exécuté durant un temps limité, jusqu'à son achèvement. Ce type d'exploitation est très lié au travail en mode dialogué. (Dictionnaire Larousse)

Ce mode d'exploitation donne l'impression à chaque utilisateur d'être le seul à utiliser la machine.

> *La commande ps aux (man ps) vous permet de voir les programmes qui tournent sur un système Unix*

Le gestionnaire de tâches de Windows vous en donne également une image

2.9 Ressources

Le deuxième rôle fondamental d'un système d'exploitation après celui de montrer une machine virtuelle est le rôle de **gestionnaire de ressources** qui apparaît avec la multiprogrammation. Le système est ici chargé de l'attribution des différentes ressources aux processus**.** Maintenant que plusieurs processus peuvent s'exécuter de manière entrelacée, se pose le problème de leur allouer les ressources de l'ordinateur tout en évitant les conflits.

Nous avons déjà parlé brièvement de la mémoire et du temps processeur, mais prenons aussi le problème classique d'une imprimante.

Imaginons ce qui se produit si trois processus qui s'exécutent « en même temps » essayent tous d'envoyer leurs résultats à l'impression : on risque de trouver sur le listing de sortie un mélange des lignes de l'un et de l'autre, ce qui est loin d'être le but recherché.

Le système d'exploitation peut éviter cela en envoyant les résultats dans un fichier tampon sur le disque (un par processus/impression) ce fichier sera déposé dans un répertoire dédié : le répertoire de spoole . Lorsque une impression « se termine », le Système peut réellement envoyer à l'imprimante le fichier en entier. Les processus croient donc envoyer leurs données à l'imprimante, mais celles-ci sont interceptées par le Système d'exploitation et mises en attente.

2.9.1 Interblocages

Les ordinateurs ont plusieurs ressources à gérer, certaines d'entre elles ne peuvent servir qu'a un processus à la fois, c'est le cas de l'imprimante, des graveurs, et même de l'accès aux fichiers en écriture. Imaginez que deux processus qui s'exécutent « en même temps » écrivent (35) dans un même fichier cela résulterait en un grand chaos (36).
Ce genre de ressource est qualifiée de « **non partageable** » .

Une deuxième caractéristique des ressources qui nous intéressent est le fait de ne pas pouvoir être retirées à un processus qui les ont acquises autrement que par décision du processus même. De telles ressources sont qualifiées de « **non préemptibles**»

Il arrive parfois que les processus aient besoin de plus d'une ressource, non partageable et non préemptible, en même temps comme dans l'exemple suivant.

Imaginons que deux processus doivent utiliser à la fois un scanneur et un graveur de CD pour y graver l'image numérisée. Soit la séquence suivante :

Le premier processus A demande le scanneur pour commencer sa numérisation d'image et l'obtient

1. Le second processus B commence par s'allouer le graveur pour y écrire des informations préliminaires.
2. C'est alors que A demande le graveur (occupé), A est donc mis en attente que B ait terminé d'utiliser le graveur.

35 Nous ne pouvons pas dire la même chose pour des lectures, pourquoi interdirait-on plusieurs processus de lire les mêmes informations au même moment ?

36 Il est à remarquer que le système Unix permet cet accès simultané par défaut.

3. Mais B continue son exécution par la demande du scanneur, il est donc mis en attente car celui-ci est utilisé par A.

Nous aboutissons à une situation où A et B sont éternellement bloqués l'un par l'autre. Cette situation qui risque de se présenter dans le cas

1. où les ressources sont non partageables et non préemptibles.
2. si plus d'une ressource est requise simultanément par plus d'un processus

se dénomme « **deadlock** » ou « étreinte mortelle » ou encore « **interblocage** » et est une des problématiques auxquelles un système d'exploitation peut apporter une solution.

Différentes stratégies peuvent être adoptées (ignorer, éviter, prévenir, détecter/récupérer...37). Il est intéressant de savoir que bon nombre de systèmes d'exploitations actuels se contentent d'ignorer de telles situations (38).

2.10 Les Systèmes d'exploitation aujourd'hui

Il existe aujourd'hui une série de Systèmes d'exploitation d'utilisation générale, en plus d'une variété de systèmes d'exploitation plus où moins orientés vers l'un ou l'autre domaine d'utilisation, parmi ces derniers :

les systèmes pour mainframes

reconnus pour leur capacité à gérer un grand nombre de programmes faisant chacun de nombreuses entrées-sorties. On les retrouve aussi aujourd'hui dans le rôle de serveurs Web et commerce électronique (OS 390, ...).

les systèmes serveur

servent en parallèle de nombreux utilisateurs à travers le réseau et permettent à ces derniers de partager des ressources matérielles et logicielles (imprimantes, disques, programmes applicatifs...) (Unix, Windows2000/3, linux,...)

les systèmes personnels

souvent dédiés à des applications bureautiques et accès à internet (Windows 98, Windows 2000, MacOs, linux,...)

les systèmes temps réel

Se caractérisent par le respect de contraintes temporelles (applications dans le monde industriel où l'instant où une action se déroule a de l'importance : chaîne de montage) (VxWorks, QNX, LynxOS, RTLinux,...)

37 Nous donnons des exemples de ces techniques au cours

38 Nous vous renvoyons au livre de Tanenbaum pour plus de détails.

les systèmes embarqués

ordinateurs qui rendent des services simples « sur mesure ». Un système d'exploitation simplifié suffit , on peut se passer de multiprogrammation et donc de la surcharge de code système qu'elle implique (PalmOS, WindowsCE, linux...)

les systèmes pour smart card

les systèmes pour multimédia

qui doivent répondre à de nouvelles exigences en terme de stockage de fichiers, mais aussi en termes d'ordonnancement temps réel

les systèmes multiprocesseur

recourent à plusieurs processeurs sur la même plate forme (systèmes d'exploitation spéciaux : variantes de systèmes serveur, améliorées au niveau de la connectivité)

les systèmes distribués

...

Lorsque on voit cette différenciation, il est dès lors évident que nous n'avons fait que lever un coin du voile sur la liste des problématiques qu'un système d'exploitation est amené à résoudre, en voici une synthèse. Dans la suite de ce cours et dans le cours des années successives nous allons en explorer quelques-unes plus dans le détail.

gestion des ressources

mécanismes de contrôle d'accès, sémaphores, gestion des interblocages

gestion des processus

problématique de l'ordonnancement, mécanismes de synchronisation,...

gestion de la mémoire

structure, mémoire virtuelle,...

entrés / sorties

systèmes de fichiers

organisation des données sur un disque, gestion des droits d'accès...

sécurité

à l'heure de l'interconnexion des réseaux par internet, cette problématique devient incontournable

...

2.11 Testez votre compréhension

1. Expliquez les avantages apportés par le traitement par lots (batch)

2. Expliquez le principe de la Multiprogrammation
3. En quoi consiste le « time slicing » ?
4. Expliquez ce qu'est un appel système et pourquoi il utilise le mécanisme des interruptions.
5. Expliquez les différents états d'un processus et illustrez les changements d'état possibles.
6. Dans quelle situation risque-t-on un interblocage ? Type de ressources, nombre, ...

3 Systèmes De Fichiers

Après avoir différencié la vue qu'un utilisateur a d'un système de fichiers de celle qu'en a le système d'exploitation, nous parlerons de la manière d'allouer l'espace disque aux fichiers (contigüe ou en blocs reliés) et pour chaque choix nous présenterons avantages et inconvénients. Nous détaillerons deux cas de systèmes de fichiers. Le plus ancien, la FAT, encore utilisé de nos jours malgré son age et NTFS plus récent, intégrant des concepts plus évolués adaptés à des environnements « multi-utilisateur » (notion de propriétaire, droits d'accès, cryptage, compression,...). Nous examinerons leur structure, la représentation de fichiers et de répertoires tout en essayant de pointer du doigt les limites de ces systèmes de fichiers et leurs causes.

Toute application informatique peut être vue comme un processus de transformation d'information. L'information en entrée est traitée pour produire une information en sortie

L'information nécessite d'être stockée sur un support ; la mémoire centrale est un premier dispositif de stockage de l'information à accès rapide, mais, elle a certains défauts : premièrement elle est volatile, son contenu est perdu si on coupe l'alimentation de l'ordinateur, en cas de panne, par exemple, on perd toute l'information en mémoire, de plus sa taille est limitée par la taille physique de la mémoire (39).

Dans le contexte d'applications de gestion de personnel comme une application qui établirait des fiches de paye d'employés, on imagine aisément avoir besoin d'un fichier décrivant les employés (salaire base, ancienneté...). Sachant que dans le cadre de certaines sociétés ou administrations, le nombre d'employés peut facilement dépasser le millier, nous pouvons difficilement nous passer d'un support de stockage « permanent » pour mémoriser ce fichier. Imaginez-vous la perte d'énergie si à la moindre panne, on devait ré encoder la liste des employés (sans compter le nombre d'erreurs d'encodage)! Pour pallier à cela, nous utiliserons des mémoires auxiliaires (disque, bande,...) ou autre support périphérique qui offrent la possibilité de mémorisation permanente de l'information. Ces fichiers peuvent être crées, détruits, lus ou écrits par les processus (applications, commandes système...).

Stockage permanent veut dire que si l'on éteint l'ordinateur et le rallume, les informations ne sont pas perdues.

Un des rôles d'un système d'exploitation est de mettre à la disposition des utilisateurs et programmeurs d'applications, un système de fichiers qui

39 Dans les systèmes à « mémoire virtuelle » cette taille n'est pas limitée à la taille physique de la mémoire (chapitre sur la gestion de la mémoire en deuxième année SYS2).

- permet de stocker des fichiers de très grande taille
- permet de stocker les informations de manière permanente
- permet à plusieurs processus d'accéder aux mêmes informations

Dans l'étude d'un système de fichiers on distingue deux approches :

1. celle de l'utilisateur qui ne voit que l'interface au système de fichiers (noms admis, accès, propriétés, permissions..),
2. celle du concepteur et programmeur système qui s'intéresse en plus, au détail de la mise en œuvre d'un système de fichiers, ici on se soucie de comment est organisé l'espace disque, comment y est répartie physiquement l'information, comment on mémorise les emplacements libres sur le disque, comment trouver à partir d'un nom de fichier son emplacement physique sur le disque, où sont décrits les attributs du fichier (protection, propriétaire, taille, date de modification...).

C'est surtout cette dernière approche qui nous intéresse dans le cadre de ce cours.

Un système d'exploitation peut reconnaître plus d'un système de fichiers (40), c'est le cas de linux, qui en plus de reconnaître différents systèmes de fichiers natifs (EXT, EXT2, EXT3) reconnaît une partition FAT et NTFS propres à DOS et Windows.
Dans le cadre de ce cours, nous nous intéresserons en particulier à deux systèmes de fichiers reconnus par le système d'exploitation Windows : FAT et NTFS. En deuxième année vous étudierez d'autres systèmes de fichiers propres au système d'exploitation Linux : EXT et EXT2.

3.1 Système de fichiers : vue utilisateur

noms de fichiers

Un processus qui crée un fichier lui attribue un nom, et lorsque celui-ci se termine le fichier persiste et un autre processus pourra l' accéder via son nom. Chaque système de fichiers a ses propres règles pour nommer les fichiers, la table suivante en montre quelques exemples.

	NTFS pour Windows (NT – 2000 – XP)	***EXT2 pour Linux***	***pour Os 390***
Noms autorisés	1 à 255 caractères Unicode excepté les caractères « \ / : * ? ” < >| » les noms « . » et « .. » interdits	1 à 256 caractères, sensible à la casse (astubu ≠ Astubu) (lettres, chiffres, « . - _ ») autorisés « ? ' ' $ & [ ] { } » non recommandés « / » interdit (séparateur) les noms « . » et « .. » interdits	Max 44 caractères en blocs de maximum 8 caractères séparés par des « . » « `ANDR.RESIDENT.CHZ.TRUCS` »

Conventions pour nommer les fichiers

Certains systèmes de fichiers gèrent le nom de fichiers en deux parties, c'est le cas de FAT16 en

40 Plusieurs systèmes de fichiers peuvent coexister sur un même disque dans des partitions différentes.

DOS qui prévoit maximum 8 caractères suivis éventuellement d'un point et d'un suffixe de maximum trois caractères supplémentaires indiquant le type du fichier (.txt, .zip, .bat, .bin, .c, ...).

Structure des fichiers

Les fichiers peuvent être structurés logiquement de différentes manières, en voici trois illustrations :

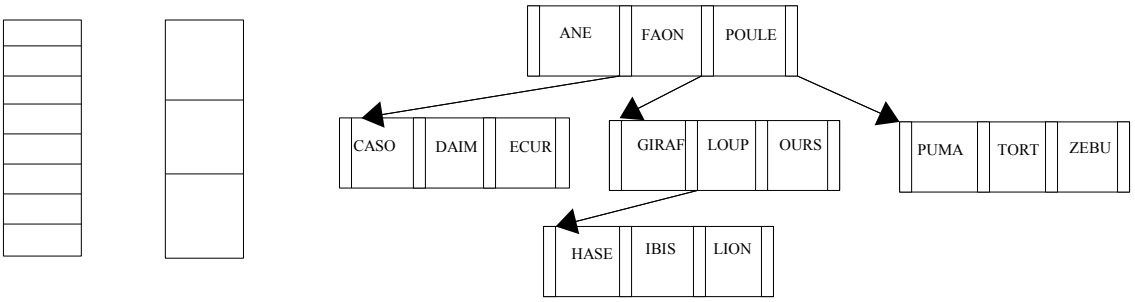


fig 3.1.1.1.1 :suite de bytes, suite d'enregistrements, arbre [TNB]

enregistrement :

ensemble d'informations traitées en bloc lors de l'échange entre différentes unités d'un ordinateur ou lors d'un traitement au sein d'un programme

Dans le cas de la suite de bytes, la structure du contenu du fichier n'est connue que par le programme d'application.

Dans le cas de la suite d'enregistrements de même taille, l'enregistrement est l'unité de lecture et d'écriture des données, la structure interne de chaque enregistrement est connue par le programme applicatif. En sont des exemples les répertoires du système FAT et le fichier MFT de NTFS.

Dans le cas d'une structure en arbre chaque enregistrement est associé à une clé et la taille des enregistrements peut être différente. L'arbre est ordonné, les recherches se font par la clé. Cette organisation permet de retrouver très rapidement un enregistrement sur le disque.

Nous aurons essentiellement à faire à des fichiers du premier type.

Les types de fichiers

Sous le nom fichier on regroupe des informations de natures parfois très différentes : fichier ordinaire, fichier catalogue (répertoire), fichier lien (lien software),...

L'accès

On distingue l'accès séquentiel de l'accès aléatoire, le premier permet d'accéder à une donnée du fichier seulement après avoir accédé aux données qui précèdent son emplacement dans le fichier (bandes magnétiques). En revanche, l'accès aléatoire qui a vu le jour avec l'arrivée des disques permet l'accès direct à une information.

Les attributs

En plus de son nom et de ses données, un fichier a toute une série d'attributs, la liste pouvant différer d'un système à l'autre. Ceux-ci sont liés à la sécurité, robustesse et autres caractéristiques d'un système de fichiers qui en font la qualité. En voici quelques exemples :

protection d'accès, mot de passe, créateur, propriétaire, indicateur de lecture seule, indicateur fichier caché, indicateur fichier ascii/binaire, date de création, date de modification, taille,...

Les opérations

Toute opération autorisée sur les fichiers est liée à l'ensemble des instructions étendues ou **appels système** mis à disposition du programmeur comme interface au système de fichiers. Parmi celles-ci on trouve généralement des ordres permettant de

- créer ou supprimer un fichier d'un répertoire (écrire dans le répertoire)
- créer ou supprimer un répertoire
- lire ou écrire dans un fichier
- positionner les attributs du fichier (modifier les droits d'accès, renommer un fichier...)
- ...

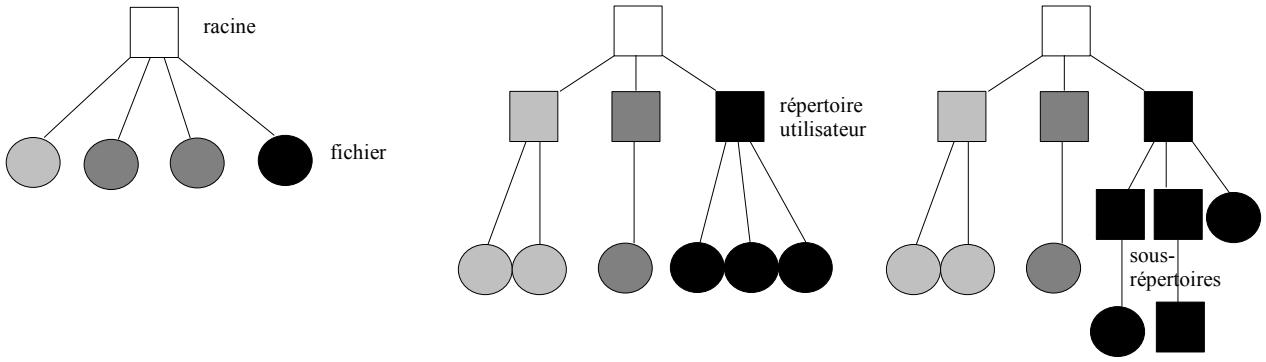
Les catalogues

Cette partie concerne l'organisation logique de l'information au sein du système de fichiers même. Le système organise le nom, les attributs des fichiers et leurs adresses dans des catalogues, qui sont souvent eux-mêmes des fichiers. L'organisation de ces catalogues diffère d'un système à l'autre.

L'organisation la plus élémentaire est l'organisation à un niveau de répertoire; elle consiste à regrouper tous les fichiers dans un catalogue unique avec le risque d'avoir des conflits de nom (si plusieurs utilisateurs décident de nommer un fichier de la même manière). Cette organisation ne convient pas pour des systèmes d'exploitation évolués (chaos).

Une amélioration de cette organisation est l'organisation à deux niveaux de répertoire qui consiste à regrouper les fichiers de chaque utilisateur dans un catalogue propre à ce dernier, celle-ci a l'avantage de particulariser les noms de chaque fichier : les fichiers nommés user1.fichier et user2.fichier seront bien deux fichiers distincts appartenant respectivement aux catalogues user1 et user2. Cette organisation ressemble à celle qu'on trouve sur OS390.

Une troisième option ; l'organisation hiérarchique consiste à introduire une arborescence dans la structure des catalogues, un catalogue pouvant à son tour en contenir un autre. Cette option permet une meilleure organisation logique de l'information. C'est ainsi que tous les fichiers images se trouveront dans un catalogue nommé image au sein duquel les images pourront être reparties par année dans des sous catalogues reprenant le nom de l'année etc. C'est l'organisation choisie par Unix, DOS, Windows et linux.



répertoire à un niveau (3 propriétaires blanc, gris noir)

répertoire à deux niveaux

répertoire hiérarchique

fig 3.1.1.1.2 :

fig 3.1.1.1.3 :les trois organisations des répertoires [TNB]

Les deux dernières organisations permettent d'avoir des fichiers différents qui portent le même nom, pour autant qu'ils se trouvent dans des répertoires différents. Pour différencier de tels fichiers on les nommera en préfixant leur nom par le chemin pour les trouver. Quand ce chemin est exprimé explicitement à partir du répertoire racine de l'arbre on dira qu'il s'agit du « chemin absolu » du fichier.

Exemples de chemins absolus :

c:\MesDocuments\adresses	sous Windows (séparateur \)
/home/mba/distri/hello	sous linux (séparateur /)

en Windows 2000 vous pouvez visualiser l'arborescence d'une partition ou répertoire dans l'explorateur, mais aussi dans la console de commandes Windows 2000 par la commande tree (« help tree »)
En linux vous pouvez visualiser l'entièreté d'une arborescence par la commande ls avec l'option -R (« man ls »)

3.2 Système de fichiers : mise en œuvre

Comme nous avons déjà évoqué plus haut, un système de fichiers doit répondre à une série de problématiques, la première d'entre elles est la manière dont l'espace disque sera alloué aux fichiers.

3.2.1 Techniques d'allocation de l'espace disque pour un fichier

3.2.1.1 allocation contiguë

Chaque fichier est d'un seul tenant sur le disque, ce qui en facilite l'exploitation.

fig 3.2.1.1.1 :état d'allocation des blocs au début avec 6 fichiers [TNB]

Toutefois cette technique d'allocation alourdit considérablement la gestion de l'espace disque. En effet, le système de fichiers doit permettre la création, suppression et extension de fichiers et sera vite confronté au problème de la fragmentation externe.

fig 3.2.1.1.2 :état d'allocation des blocs après suppression des fichiers D et F [TNB]

Lorsque l'espace disque est fort fragmenté il vient à manquer l'espace pour stocker de nouveaux grands fichiers. Cette technique oblige à réaliser des déplacements de fichiers pour faire de la place sur le disque et provoque des délais d'attente.

> **fragmentation externe** :
>
> l'espace libre sur le disque est fragmenté : concerne les zones libres du disque qui sont en dehors des fichiers

3.2.1.2 allocation par blocs

Chainage sur base d'un index

L'unité d'allocation d'espace disque est un bloc de taille fixe (par exemple 4Kb). Chaque bloc peut être adressé par son numéro d'ordre (41). Un fichier est enregistré dans une série de blocs sur le disque. Dans cette technique d'allocation, l'emplacement des blocs d'un même fichier n'est pas obligatoirement consécutif. Nous avons besoin d'informations supplémentaires par fichier pour en retrouver l'entièreté des données. Ces informations se trouvant bien sûr également sur le disque.

41 Si le bloc 0 commence au secteur 0 du disque, alors le bloc N commence au secteur N x Nombre de secteurs par bloc

Un fichier occupe un nombre entier de blocs, tout fichier, même si il contient un seul byte, occupera au moins un bloc sur le disque. Dans ce cas un byte du cluster sera alloué au fichier et le reste sera «inutilisé». Le dernier bloc de données d'un fichier peut être partiellement inutilisé, l'espace perdu dans ce dernier bloc se nomme « **fragmentation interne** ».

C'est ainsi qu'un fichier se voit associé deux tailles : la taille logique (le nombre de bytes utiles) et la taille physique (le nombre de bytes réellement occupés sur le disque par les données : multiple entier de la taille d'un bloc).

> *la commande « ls -l » sur unix permet de voir la taille logique d'un fichier alors que la commande du (disk usage) montre la place réellement allouée sur le disque pour le fichier*
> *Windows 2000 permet de voir les deux tailles dans les propriétés du fichier. La commande dir montre la taille logique*

L'entièreté du fichier est reconstituée par un index : table (42) qui donne la séquence de blocs constituant le fichier : chaque entrée de la table donne le numéro du bloc suivant

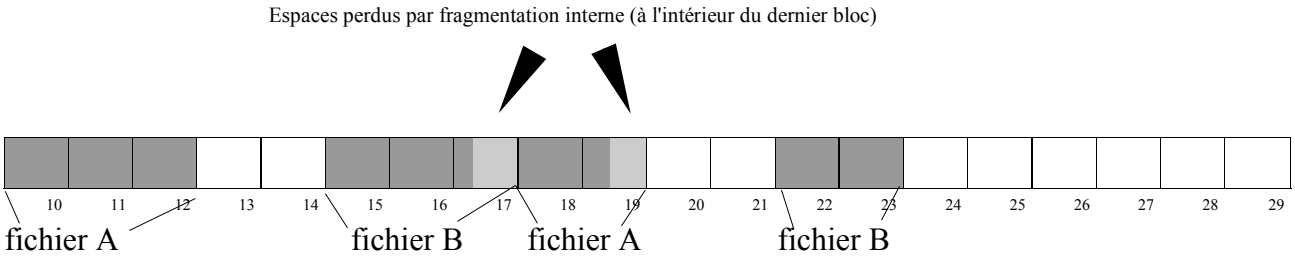


fig 3.2.1.2.1 :blocs sur le disque

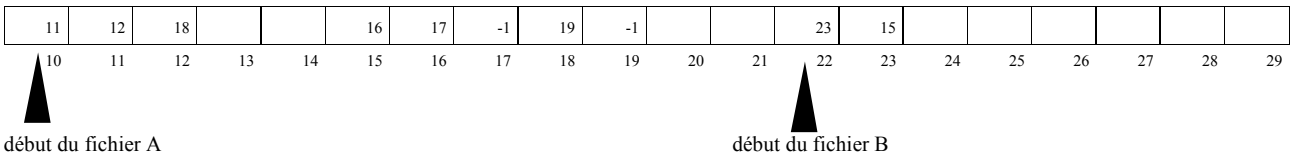


fig 3.2.1.2.2 : chainage des blocs par table d'index

Dans le dessin qui précède, nous avons représenté en haut les blocs du disque avec les données des fichiers en noir et en bas la portion de table d'index correspondante avec une entrée par bloc du disque. A partir de cette table nous pouvons reconstituer la séquence des blocs appartenant à un fichier à condition de connaître le **numéro du premier bloc** de celui-ci qui sera lu dans le répertoire parent. Dans l'exemple, le fichier A a pour premier bloc le bloc 10 et B le bloc 22.

Chaque bloc est chainé au suivant dans la table d'index dans l'entrée correspondant à sa position, le

42 Nous voici donc confrontés à la nécessité de représenter un tableau , cette structure est fréquemment utilisée en informatique et vous y serez très vite familiarisé dans vos cours de programmation. Un tableau permet de regrouper plusieurs informations de même structure sous un seul objet en mémoire (vive ou auxiliaire), chaque information pouvant être obtenue moyennant son indice : sa position dans le tableau.

suivant du bloc 10 en position 10 etc. Chaque entrée de la table contient le numéro du bloc suivant qui n'est pas toujours le suivant sur le disque. Dans notre cas le bloc 10 est suivi par le 11. Le fichier A correspond donc à la séquence de blocs : 10 11 12 18 19, le fichier B à : 22 23 15 16 17. Les valeurs -1 en position 19 et 17 de la table d'index indiquent que ce bloc est le dernier d'un fichier.

Étant donnés les blocs d'un disque, localisés par leur numéro, et la table d'index dont l'emplacement sur le disque est connu, nous pouvons reconstruire l'entièreté d'un fichier en connaissant le numéro de bloc de début (10 pour A et 22 pour B). Nous avons déjà dit que le numéro du premier bloc d'un fichier se trouve dans la description du répertoire parent. Cette description sera vue en détail plus loin.

Le problème de la fragmentation externe a moins de conséquences, mais apparaît celui de la « **fragmentation interne** » : perte de place à l'intérieur du dernier bloc qui n'est pas toujours entièrement rempli. Le choix de la taille des blocs va influencer la performance du système de gestion : une grande taille demandant moins d'accès disque (lecture/écriture séquentielles du fichier plus rapides) donnera toutefois une fragmentation interne plus importante donc une moins bonne utilisation de l'espace disque.

C'est cette méthode d'allocation qu'a choisi le système de fichiers FAT. FAT signifie File Allocation Table qui est le nom donné à sa table d'index.

> Fragmentation interne :
>
> concerne les zones du disque « libres » perdues à l'intérieur du dernier bloc alloué aux fichiers.

Chaînage par listes de blocs

Il s'agit d'une autre manière de chainer les blocs de taille fixe, elle n'utilise pas de table d'index pour reconstituer les fichiers. Ici les blocs sont chaînés entre eux : chaque bloc contiendra une information de chaînage : le numéro du bloc suivant du fichier. Le dernier bloc contenant une « marque de fin de fichier ».

L'exemple suivant montre un fichier qui occupe dans l'ordre , les blocs : 4, 7, 2, 10, 12

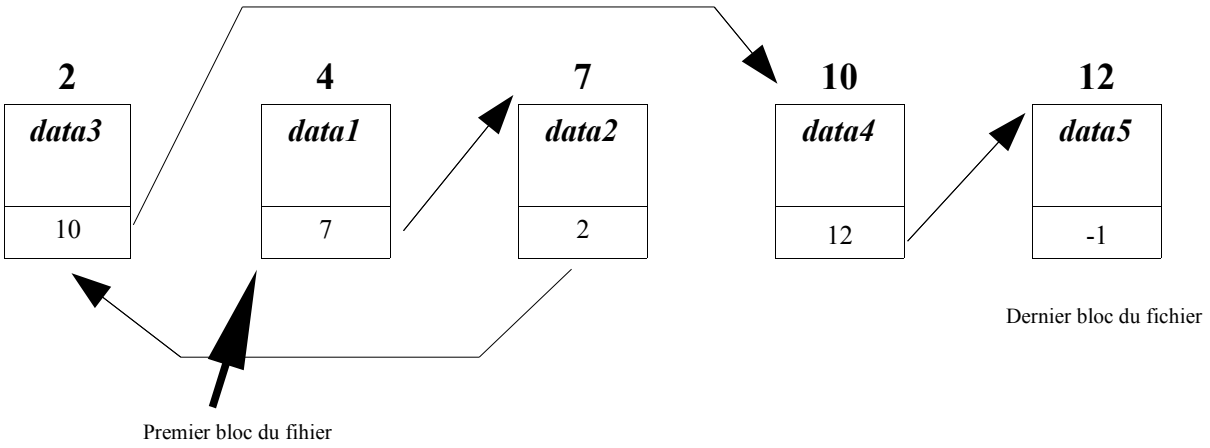


fig 3.2.1.2.3 :un fichier représenté par des blocs chaînés

Pour accéder une donnée qui se trouve dans le troisième bloc du fichier ('data3'), nous avons besoin

de connaître son emplacement et donc lire les blocs data1 et data2.

Comme au paravent, nous devons connaître le début : l'adresse du premier bloc, dans celui-ci nous lisons l'adresse du bloc suivant.

L'avantage de cette technique est qu'on évite de stocker un index sur le disque , mais on perd l'accès direct aux données : l'accès au nième bloc du fichier nécessite de parcourir toute la chaîne et donc d'accéder les n-1 blocs qui le précèdent.

Si les méthodes d'allocation contiguë présentent un problème de fragmentation externe, les méthodes d'allocation par blocs, outre un problème de fragmentation interne sont sujettes à avoir des **fichiers fort fragmentés** c'est à dire avec des blocs fort éparpillés. Accéder aux données demande alors beaucoup de déplacements de têtes de lecture, ce qui ralentit.

Les systèmes de fichiers FAT et NTFS ne gèrent pas eux-mêmes la **défragmentation des fichiers**. Toutefois des utilitaires de défragmentation existent et peuvent être exécutés sur demande ou de manière automatique.

3.3 FAT

Le système de fichiers FAT (File Allocation Table) est né avec le système MS-DOS et est utilisé encore de nos jours sur les systèmes d'exploitation Windows bien que NTFS lui ait succédé.

FAT, comme NTFS, divise l'espace disque en unités d'allocation appelées « **clusters** », un cluster est un ensemble contigu de secteurs, de taille fixe (le plus souvent 512 bytes). Quand un fichier est créé ou modifié, l'espace disque lui est donc alloué/desalloué par blocs (clusters).

cluster :

plus petit espace contigu sur le disque, pouvant être alloué pour y stocker un fichier.
Unité d'allocation.

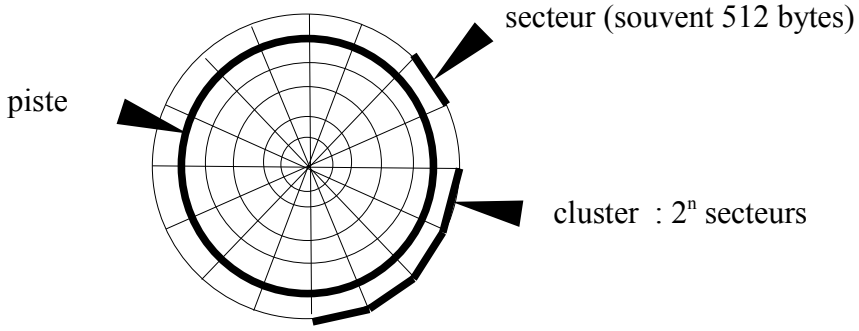


fig 3.3.1.1.1 : piste, cluster, secteur

La taille du « cluster » est fixe et est un multiple de la taille d'un secteur (taille = 512 bytes) (N x 512 avec N = puissance de 2 : 512, 1Kb, 2Kb, 4Kb,..) . La taille des clusters d'une partition est fixée

au moment du formatage de celle-ci.

Si la taille d'un cluster est *tc*, l'espace perdu par fragmentation interne par fichier est en moyenne *tc*/2.

Quand un fichier utilise un espace supérieur à un cluster, on lui alloue plusieurs clusters non nécessairement contigus. Le fichier est reconstitué à l'aide d'une table de chaînage des clusters globale (une pour toute la partition) appelée FAT (File allocation table) (43). Cette dernière permet de reconstituer chaque fichier en entier, une fois que le premier cluster du fichier est connu.
La structure interne de FAT correspond à ce que nous avons illustré plus haut en 3.2.1.2.

Nous avons la clé pour retrouver le contenu d'un fichier une fois que nous connaissons le numéro du premier cluster.

Si il est assez intuitif de comprendre que les données d'un fichier se trouvent dans les clusters composant un fichier il est moins intuitif de comprendre où sont mémorisés les attributs des fichiers (nom, date de création, permissions d'accès...) et comment sont représentés les répertoires. Le système de fichiers FAT utilise une organisation de fichiers arborescente, une partition possède un répertoire racine qui contient des fichiers et des sous-répertoires, ces derniers pouvant, à leur tour, contenir d'autres fichiers et sous-répertoires etc. Voyons plus en détail comment cette organisation est représentée en FAT.

3.3.2 Structure d'un système de fichiers FAT

3.3.2.1 zones

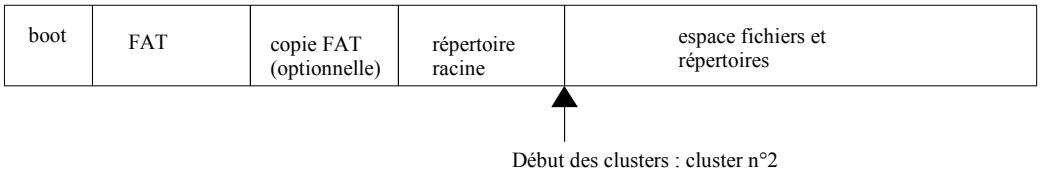


fig 3.3.2.1.1 :un système de fichiers FAT16

3.3.2.2 secteur de boot ou zone réservée

Le secteur de Boot (boot record) est une zone réservée au système d'exploitation, elle contient notamment le code exécutable permettant de charger le système d'exploitation en mémoire. Cette zone a une taille de 1 secteur (secteur 0) en FAT12 et FAT16, elle a une taille de 32 secteurs en FAT32. Le boot record contient également des données très importantes qui permettent d'interpréter correctement toutes les informations contenues dans la partition. Parmi celles-ci :

- TS - la Taille d'un Secteur (**512**, 1024, 2048, ou 4096)
- NSC - le Nombre de Secteurs par Cluster (1, 2, 4, 8, ... 64,128)
- TZR - la Taille de la Zone Réservée (boot record) (1 ou 32)
- NERR - le Nombre d'Entrées dans le Répertoire Racine (sur 2 bytes) (il s'agit du nombre **maximum** d'entrées en FAT12 et FAT16, cette donnée vaut 0 en FAT32)

43 FAT indique donc à la fois le système de fichiers et la table d'index qu'il utilise pour le chaînage des clusters. Pour éviter toute confusion, nos appellerons « FAT » le système de fichiers et « la FAT » sa table d'index.

- NS - le Nombre de Secteurs de la partition
- NF - le Nombre de tables FAT
- NSF - le Nombre de Secteurs de la FAT
- ...

Sachant que le **Cluster n°2**, par convention, démarre au répertoire racine pour une FAT32 et juste après ce dernier pour une FAT12/16 , et que chaque entrée du répertoire racine occupe 32 bytes, il est possible au système d'exploitation de calculer avec précision la position (n° de secteur) du cluster n° N.

3.3.2.3 la FAT et sa copie

La FAT (File Allocation Table), table d'allocation de fichiers mémorise l'utilisation de l'espace disque. Une FAT a une entrée pour chaque cluster du disque, ses entrées ont une taille fixe (12 bits pour les premières versions de DOS (FAT12), 16 pour une FAT16 et 32 pour une FAT32). Chaque entrée contient le numéro de cluster suivant pour le fichier. Pour une FAT 32 chaque entrée est stockée sur 32 bits, mais seulement 28 sont utilisés.

Dans une FAT12 chaque entrée est codée sur 12 bits, on peut donc avoir maximum 2^{12} clusters numérotés de 0 à 2^{12}-1. Si les clusters ont une taille de 512 bytes, la taille d'une partition formatée en FAT12 est donc limitée à 2Mb (2^{12} X 512 bytes). On peut évidemment augmenter cette taille maximum en choisissant des clusters plus grands : avec des clusters de 16Kb, une FAT 12 pourrait atteindre 64 Mb (2^{12} X 16 Kb), dommage toutefois pour l'espace perdu par fragmentation interne, dans ce cas le plus petit fichier occuperait 64 Kb sur le disque.

Le même raisonnement s'applique aux FAT 16 (16 bits) et 32 (28 bits). Avec des clusters de 512 bytes nous obtenons 32 Mb comme taille maximum pour une partition en FAT16, 128 Gb pour une FAT32.

On a vu que la taille des clusters peut être configurée au moment de formater une partition (44), voici donc quelques configurations possibles d'une FAT16 en fonction de sa taille (45) :

Taille de la partition	***Taille des Clusters***
< 32 Mb	512 bytes
32Mb <= x < 64 Mb	1 Kb
64Mb <= x < 128 Mb	2 Kb
128 Mb <= x < 256 Mb	4 Kb
...	...
2 Gb	32Kb

fig 3.3.2.3.1 :

fig 3.3.2.3.2 :FAT16 : taille des clusters et des partitions

Comme le montre le tableau ci-dessus, en FAT16, pour des grandes partitions, la taille des clusters devient vite très grande et les pertes d'espace par fragmentation interne sont très importantes.

Les numéros de clusters pouvant être attribués sont dans une plage plus restreinte que l'ensemble

44 Toutefois, avec des secteurs de 512 bytes, la taille d'un cluster est limitée à 64 Kb. Pour des secteurs plus grands les clusters peuvent avoir 128Kb voir même 256 Kb comme taille.

45 Il est à remarquer que au delà des limitations théoriques, FAT16 et FAT32 imposent certaines restrictions sur le nombre de clusters pour une partition,

des numéros qu'il est possible de représenter sur 16 bits. Il existe en effet quelques valeurs réservées avec une signification particulière : cluster inutilisé, fin d'un fichier, cluster défectueux.

Les entrées de la FAT contiennent une des valeurs suivantes :

Valeur hexadécimale et décimale	*Signification*
(0)000H -/- 0	Cluster disponible
(F)FF0-(F)FF6H -/- 65520-65526	Cluster réservé à MS-DOS
(F)FF7H -/- 65527	Cluster détruit; rayé de FAT (erreur d'accès)
(F)FF8-(F)FFFH -/- 65528-65535	Dernier cluster alloué pour le fichier (fin de fichier)
(X)XXX	Toutes les autres valeurs : numéro relatif du prochain cluster alloué pour ce fichier

fig 3.3.2.3.3 :

fig 3.3.2.3.4 :FAT12 et FAT16 : valeurs particulières des entrées

Pour une FAT 16 la plage est (0)001h à (F)FEFh . Le 0 (**cluster libre** et non pas chaîné au cluster 0) et les 16 derniers numéros ont un usage réservé.

Le système a besoin de reconnaître les clusters libres lorsque un fichier augmente de taille, quel cluster lui allouer ? Ce sera l'appel système qui permet d'écrire dans un fichier qui se chargera de cette nouvelle allocation.

Une copie de la FAT suit la FAT assurant une meilleure résistance aux pannes. En effet si la FAT est endommagée (secteur défectueux), de nombreux fichiers ne peuvent être reconstitués.

3.3.2.4 méta-données et répertoires

En FAT, chaque fichier ou répertoire est décrit par un descripteur de 32 bytes. Ces descripteurs se trouvent dans les données du fichier répertoire parent. Un répertoire est un fichier particulier et ses données ne sont autres qu'une liste de descripteurs de 32 bytes, un par fichier ou sous-répertoire. Chaque descripteur de 32 bytes contient le nom (46) du fichier, ses attributs : date, numéro du premier cluster, taille, les bits d'attribut...

> *VFAT et FAT32 supportent les noms de fichiers longs. Ceux-ci sont stockés dans un dossier spécial d'enregistrement avec la configuration des attributs Lecture Seule, Fichier caché, Système et Volume. Cette particularité est née avec VFAT (Virtual FAT), mais cet aspect ne change pas fondamentalement la structure de FAT.*

46 8 caractères + 3 pour l'extension : limitation valable uniquement en FAT12 et en FAT16

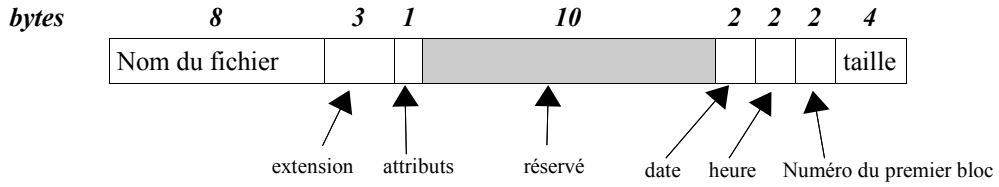


fig 3.3.2.4.1 : les 32 bytes d'une entrée de répertoire en FAT16

Bits d'attribut :

A : fichier modifié, non sauvegardé
D : répertoire
V : nom d'un volume
S : fichier système (non effaçable)
H : fichier caché
R :lecture seule

dans la console de commandes Windows 2000 vous pouvez visualiser certains attributs d'un fichier par la commande attrib (« help attrib»), d'autres par la commande dir. La commande dir /A permet de lister les fichiers selon leurs attributs.
La commande attrib permet aussi de positionner certains attributs

Date (16 bits structurés) :

jour du mois : 5 bits
mois de l'année : 4 bits
année depuis 1980 : 7 bits (1980 + 127 ans -> 2107)

Heure (16 bits structurés) :

secondes : 5 bits (0-29 : précision de 2 secondes)
minutes : 6 bits
heures : 5 bits

Couvre une période de 127 ans depuis une date de référence (ici le 1 janvier 1980) avec la précision de deux secondes.

Cette représentation est à comparer avec celle de unix : 32 bits signés contenant le nombre de secondes depuis le 1/1/1970 . Couvre 68 ans avec une précision de 1 seconde-> (1970 + 68 ans = 2038 (47)) .

32 bits non signés auraient permis de couvrir 136 ans avec la précision de 1 seconde !

Numéro du premier bloc :

renseigne le premier bloc du fichier, les suivants sont obtenus par chaînage sur base de la table d'index.

47 En réalité la période couverte remonte à la fin de l'année 1901 ;-). En effet, en 2038 on changera de signe et on reculera de 136 ans. Le bug de l'an 2038 , unix millenium bug y2K38, est lié à la représentation de l'heure sur un entier signé en 32 bits.

Les données d'un répertoire contiennent un descripteur de 32 bytes par fichier suivant le format décrit ci-dessus.

Sachant que le fichier n'est pas lu dans sa totalité en une seule opération mais bien par plusieurs accès consécutifs, le système devra mémoriser la position de lecture pour chacun des fichiers utilisés (ouverts). Linux utilise une table en RAM : la table des descripteurs de fichiers ouverts.

3.3.2.5 répertoire racine

Le répertoire racine est un répertoire fondamental car tout chemin absolu est exprimé à partir de la racine et chaque fichier est retrouvé grâce à ce chemin absolu. Pour trouver le cluster initial de tout fichier on a besoin de lire le répertoire qui le contient. Dans le cas d'un chemin à plusieurs niveaux de hiérarchie nous serons obligés de lire tous les répertoires de la hiérarchie en partant de la racine. En effet, le répertoire contenant le fichier se trouve à un emplacement indiqué dans la description de son père, et ainsi de suite.

Comment trouve-ton le répertoire racine ? Où est son parent ?

Comme ce répertoire fait exception, dans le sens où il n'a pas de parent qui le décrit, généralement pour lui, on adopte des conventions. En FAT, ce répertoire a la particularité de se trouver à un emplacement connu (48): il suit la copie de la FAT et précède la zone de données.
En FAT12 et en FAT16, l'entièreté de ce répertoire était décrite à cet endroit. A cause de ce choix, le répertoire racine avait une taille fixe maximum et donc un nombre d'entrées (de fichiers) limité. Ce répertoire ne pouvait être étendu contrairement aux autres répertoires.

En FAT 32 cette limitation a été levée. A l'emplacement connu (cluster 2) se trouve uniquement le premier cluster du répertoire racine, la suite étant chaînée dans la FAT comme pour tout autre fichier ou répertoire.

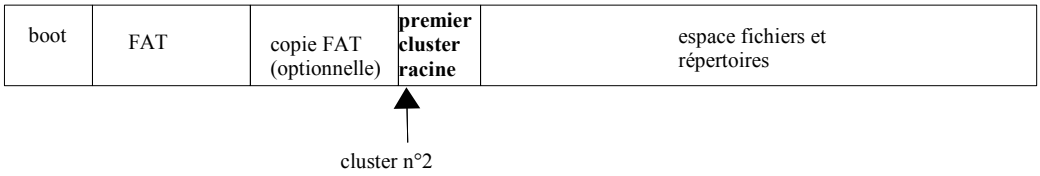


fig 3.3.2.5.1 :un système de fichiers FAT32, détail racine

3.3.2.6 espace fichiers et répertoires

Il s'agit de l'ensemble de clusters réservés aux données des fichiers et aux données des répertoires (racine exclue bien entendu). Les données d'un répertoire sont donc une suite d'entrées de 32 bytes décrivant les fichiers et sous-répertoires (comme décrit plus haut).

48 EXT utilise également une convention : la racine est décrite dans l'inode n° 2.

3.3.3 Résistance aux pannes

Tout disque n'est pas éternel et il arrive qu'un ou plusieurs secteurs d'un disque soient endommagés. Un autre ennemi redouté par les systèmes de fichiers est la panne de courant.
En FAT, malgré que les écritures des données sur le disque tout comme la mise à jour de la FAT soient faites de manière synchrone (directement, sans passer par une cache), on n'est pas à l'abri d'incohérences lors d'une panne de courant. En effet rien n'empêche une panne de survenir entre deux mises à jour de la FAT nécessaires à une même modification (comme dans le cas de suppression ou création de données). Selon le moment où intervient la panne, on pourrait se trouver avec des clusters qui sont à la fois marqués comme libres dans la FAT et chaînés à des fichiers ou des clusters marqués occupés et n'appartenant à aucun fichier (49) .

La commande en ligne « chkdsk » vérifie un disque et affiche un rapport d'état, elle permet de récupérer les clusters occupés qui ne seraient plus rattachées à aucun fichier et les transforme en fichiers nommés FILE0000.CHK, FILE0001.CHK, qui, dans le cas de fichiers au format texte peuvent être visionnés par un éditeur.
La commande recover documentée dans l'aide en ligne Windows2000 permet de récupérer (partiellement) un fichier dans le cas de secteurs endommagés.
L'utilitaire fsck est le pendant de chkdsk pour les sytèmes de fichiers linux.

3.3.4 En conclusion

Le système de fichiers FAT12 et FAT16 sont assez simples, ils convenaient pour des partitions de petite taille comme des disquettes, en effet, dès que la taille des partitions augmente, la fragmentation interne y est très importante. Pour les partitions de grande taille FAT16 et FAT12 sont remplacés pat FAT32.

Le système FAT souffre d'un défaut commun aux systèmes d'allocation par blocs (FAT, NTFS,...) : à la longue, les fichiers finissent eux aussi, par être fortement fragmentés, leur accès, même séquentiel, entraîne des déplacements fréquents des têtes de lecture et donc un ralentissement. Pour pallier à cela il est utile de faire tourner un programme de défragmentation régulièrement.

Une autre cause de ralentissement du système est l'écriture synchrone des données, il existe des systèmes de gestion de fichiers plus évolués utilisant des techniques de buffering en mémoire (mémoire cache) comme le font « ext » de Unix et NTFS de Windows.

Faites tourner l'utilitaire de défragmentation sur votre partition

Lancez la commande chkdsk dans une console avec l'interpréteur de commandes Windows (start – cmd), help chkdsk vous indique les paramètres possibles.

49 Un exemple d'incohérence du à une panne de courant sera illustré au cours.

3.4 Systèmes de fichiers avancés

Un système de fichiers plus évolué devra relever plusieurs défis en plus de satisfaire des exigences de base comme organiser l'allocation de l'espace et permettre l'utilisation de catalogues. En effet, de plus en plus, nous travaillons dans des environnements multiutilisateur, avec des données dont la perte ou la violation risquerait d'être critique. Un bon système de fichiers doit, aujourd'hui, donner des garanties de confidentialité, de résistance aux pannes et de récupération des informations en cas de panne tout en garantissant des temps de réponse acceptables. Parmi les qualités qu'un système de fichiers doit avoir de nos jours on peut citer la fiabilité, la sécurité des données et la performance.

3.4.1 Fiabilité

La destruction d'un système de fichiers est souvent beaucoup plus grave que la destruction d'un ordinateur qui lui peut être remplacé à un coût raisonnable. Protéger les informations des pannes ou erreurs logicielles est un des services fondamentaux qu'un système de fichiers évolué doit pouvoir garantir.

3.4.1.1 identification des blocs endommagés

La première implication est que le système sera amené à identifier les blocs endommagés du disque pour éviter de les réutiliser. Sur une FAT on a vu que de tels blocs sont marqués au niveau de l'index : la FAT(une valeur spéciale indique l'état défectueux).

Dans le cadre d'une allocation par blocs chaînés, on peut plus facilement envisager de créer un fichier particulier contenant la liste des blocs corrompus.

3.4.1.2 backup

La deuxième implication est que le système de fichiers doit prévoir un système de sauvegarde des données. Cette sauvegarde peut être réalisée en copiant les données sur bande ou sur un deuxième support identique régulièrement. La copie complète prendra beaucoup de temps et d'espace, proportionnellement à la taille des données à sauvegarder, elle devra être réalisée à un moment creux d'utilisation du système pour ne pas entraver son fonctionnement, par exemple toutes les nuits.

Une méthode plus économe en temps est la sauvegarde incrémentale. Une sauvegarde globale tous les mois (par exemple) intercalée de sauvegardes partielles où on sauve uniquement les fichiers ayant été modifiés depuis la dernière sauvegarde globale (ou partielle), ce qui, par ailleurs, nécessite plus de supports : 31 supports différents (bandes ou autre).

L'attribut « Archive » en FAT permet un archivage sélectif.

> *Sur Windows 2000 on peut réaliser un backup et le restaurer grâce à l'utilitaire de backup (programs–accessories–system tools-backup).*

3.4.1.3 cohérence

Troisième aspect de la fiabilité d'un système de fichiers est d'en préserver la cohérence, en effet, comme nous l'avons déjà dit plus haut, il est fréquent qu'une modification de fichier entraîne non plus une, mais plusieurs écritures sur le disque (mise à jour des données, des méta données et des informations de chaînage de la FAT...), si une panne survient avant l'achèvement de la totalité de ces opérations, le système risque de se trouver dans un état incohérent pouvant causer des redondances ou pertes d'information. Un système de fichiers doit également mettre à disposition quelques outils permettant de récupérer au mieux la situation (commande « chkdsk » en DOS-Windows, commande « fsck » en Unix). Nous avons vu que la commande chkdsk crée des fichiers avec des listes de clusters qui n'appartiendraient plus à aucun fichier et permet ainsi de raccrocher des données autrement perdues.

3.4.2 Sécurité

Il existe plusieurs points sur lesquels un système d'exploitation doit garantir une sécurité (intrusions, virus...), le chapitre sur la sécurité pourrait à lui seul faire l'objet d'un cours, nous allons approcher cette problématique partiellement.

Les ordinateurs renferment des informations qui souvent doivent rester confidentielles. Un système d'exploitation doit assurer cette confidentialité contre les accès « non autorisés ». Autre aspect de ce même volet est le partage d'information, en effet, en plus de préserver une confidentialité des données personnelles de chaque utilisateur, le système doit permettre à des utilisateurs travaillant sur un projet commun de partager des fichiers.

Nous allons explorer un des mécanismes de protection qu'un système de fichiers en particulier peut mettre en œuvre pour se protéger de tels accès : les ACL (Acces Control List).

On pourrait s'entretenir sur ce qu'est un accès non autorisé : par exemple un utilisateur qui travaille avec un login et mot de passe usurpés, est-il autorisé où non ? Ceci nous mènerait à une discussion sur l'authentification des utilisateurs et la politique de protection des mots de passe au sein d'une entreprise ou domaine informatique qui sort du cadre de ce cours. Nous n'allons donc pas envisager ce débat en écartant l'éventualité de ce genre d'abus.

3.4.2.1 domaines de protection

Chaque objet sur un ordinateur peut être identifié de manière unique et possède un jeu déterminé d'actions que certains acteurs (50) peuvent effectuer sur lui (un utilisateur peut avoir le droit de lire un fichier mais pas d'y écrire).

Un domaine est un ensemble de paires (objet, droits) où les droits définissent un ensemble d'opérations permises pour un acteur ou pour un groupe d'acteurs sur l'objet.

Par exemple Unix utilise un masque de bits pour protéger ses fichiers . Parmi ceux-ci 9 vous sont probablement déjà familiers rwx rwx rwx (« read write execute » pour le propriétaire, pour le groupe du propriétaire, pour les autres) le bit positionné à 1 autorise l'opération à l'acteur concerné. Par ailleurs, chaque acteur est identifié par un identifiant utilisateur et un identifiant du groupe auquel il appartient.

50 nous parlons explicitement d'acteur et ce terme comprend notamment des programmes qui s'exécutent en mémoire par l'initiative d'un utilisateur (ou processus).

Un processus s'exécute au sein d'un domaine de protection et celui-ci permet d'identifier les accès aux objets qui lui sont autorisés.

De cette vision sont issues deux techniques de représentation :

- les ACL - liste de contrôle d'accès (Acces Control List)
- la C-list - liste des capacités (Capability List)

Les ACL

Le principe des ACL est d'associer à chaque fichier (51) la liste des autorisations par domaine, cette technique est utilisée en EXT et en NTFS.
Aux laboratoires du cours de langage de programmation vous vous familiariserez avec une forme simplifiée d'ACL.. Cette dernière permet d'associer à un objet maximum deux domaines de sécurité : celui des utilisateurs appartenant au même groupe que le propriétaire du fichier et celui des utilisateurs extérieurs à ce groupe.

Lorsque un fichier est crée, son descripteur de sécurité aura une valeur par défaut.

NTFS sous Windows utilise également des ACL et permet plus de finesse que EXT et EXT2 dans la définition des permissions.

La commande Unix « ls » avec l'option « -l » permet de visualiser les listes de permissions sur les fichiers.
Si le résultat de la commande ls -l donne la ligne
-rw-r--- --- 1 mba prof 24 2006-10-19 22:46 monClavier
cela signifie que :
l'utilisateur mba est autorisé à lire et écrire dans le fichier monClavier
les membres du groupe prof sont autorisés à lire le fichier monClavier
les autres utilisateurs n'ont aucun droit sur le fichier monClavier

La commande Unix « chmod » permet de modifier la liste de permissions sur un fichier.
En Unix, la valeur par défaut du descripteur de sécurité est positionnée par la commande interne umask (man bash).

La commande windows CACLS affiche ou modifie les ACLs des fichiers d'une partition NTFS.
Windows permet de visualiser et modifier les ACL par l'onglet sécurité de la fenêtre « propriétés » obtenue en cliquant sur un fichier (explorateur) par le bouton droit.

La C-list

La C-list utilise une représentation inverse de celle des ACL, et associe à chaque acteur (processus) la liste des objets sur lesquels il a des permissions d'accès et la description de ces permissions. Nous

51 ce concept ne concerne pas uniquement les fichiers en réalité on parle plutôt d'objets et les fichiers sont notamment vus comme des objets.

nous étendrons pas sur cette technique.

3.4.3 Performance

Les accès disque sont de l'ordre de 1 000 000 fois plus lents que les accès mémoire. Cette différence ne fait qu'augmenter.

3.4.3.1 mémoire cache

Une technique courante pour améliorer les temps de réponse est d'utiliser une antémémoire (cache), il s'agit d'une suite de blocs qui appartiennent logiquement au disque mais sont situés physiquement en mémoire. Lorsque un bloc est utilisé et qu'il n'est pas présent en cache, il y est ramené. Toute modification du bloc a lieu dans la cache uniquement jusqu'au moment où la synchronisation des informations cache/disque est requise.

Ce système, introduit un délai plus important entre les opérations sur un fichier et leur réel aboutissement sur le disque créant une nouvelle fragilité liée aux interruptions de courant. De véritables stratégies sont liées à cette technique en vue d'en améliorer les performances tout en préservant les données au maximum.

Comment choisir le meilleur bloc à sortir de la mémoire cache lorsque celle-ci est pleine et un nouveau bloc est demandé par une opération d'entrée/sortie (en d'autre termes : quel est le bloc dont on est certains ne pas devoir se servir dans l'immédiat) ?

A quel moment synchronise-t-on les données cache/disque ? Doit-on traiter différemment blocs de données et méta données en raison de la plus grande importance de ces derniers ? ... Voici les questions auxquelles un Système de fichiers doit répondre.

Cette technique, largement utilisée sous Unix, ne sera pas développée dans le cadre de ce cours.

3.4.3.2 réduire les déplacements du bras de lecture

Une deuxième technique visant à améliorer les performances consiste à regrouper les informations susceptibles d'être accédées en séquence éventuellement de les regrouper sur un même cylindre et réduire ainsi les déplacements du bras de lecture. On peut également prendre en compte le temps de rotation du disque. Quand il alloue des blocs, le système tente de placer les blocs consécutifs d'un fichier sur un même cylindre. Ainsi, si le temps de rotation est de 16,67 ms et s'il faut environ 4ms pour qu'un utilisateur demande et obtienne un bloc, l'emplacement optimal pour le bloc suivant est ¼ de disque plus loin (52).

52 C'est bien sûr le système d'exploitation au travers des appels système (machine virtuelle) qui s'occupe de cette gestion et non le programmeur d'applications.

3.5 NTFS

3.5.1 Quelques particularités

NTFS est le système de fichiers natif de Windows NT.

Windows 2000 et Windows XP exploitent une version améliorée de NTFS : NTFS5/NTFS2000.

NTFS rompt la contrainte d'assurer une compatibilité ascendante depuis MS-DOS : alors que NT, 2000 et XP peuvent lire une partition FAT (2000 et XP même une FAT32, alors que NT ne le fait pas), la compatibilité ascendante avec les autres systèmes de Microsoft n'est pas assurée : MSDOS ne sait pas lire une partition NTFS (53).

3.5.1.1 noms longs et Unicode

NTFS offre les noms de fichiers longs, les noms de fichiers peuvent compter jusqu'à 255 caractères Unicode. L'Unicode est codé sur 16 bits et permet donc la représentation de caractères autres que ceux de l'alphabet latin.

3.5.1.2 attributs étendus

NTFS utilise un système de représentation des attributs des fichiers très souple. En effet chaque attribut a une longueur variable et le nombre d'attributs connus du système est lui-même en théorie variable.

3.5.1.3 flux de données multiples

NTFS permet d'associer plus d'un flux de données à un même fichier. Cette particularité à été introduite pour assurer une compatibilité avec Macintosh.

3.5.1.4 taille des fichiers et des partitions

Les limites de la taille des fichiers et partitions de NTFS dépassent la taille des disques actuels.

Comme les offsets (déplacements dans un fichier) sont codés sur 64 bits la taille des fichiers peut atteindre 2^{64} (16Eb Exabytes) = 17179869184 Tb (Térabytes) (pour comparaison : avec des blocs de 4Kb le système FAT 32 est limité à 2Tb et EXT de Unix (54) peut gérer des tailles de fichiers allant jusqu'à 4 Tb). De plus, une partition NTFS peut contenir jusqu'à 2^{63} clusters et chaque cluster peut mesurer jusqu'à 64Kb, ce qui fixe la limite d'une partition NTFS à 500 Trillions de Gb.
NTFS assure une meilleure capacité à évoluer vers les disques volumineux de taille très supérieure à FAT32 sans dégradation.

3.5.1.5 compression de fichiers

Depuis NTFS 3.51, NTFS supporte également la compression fichier par fichier, nous aborderons ce point plus en détail.

53 Windows NT ne sait pas lire une partition FAT 32
54 avec des blocs de 64 Kb EXT couvre des partitions de 256000 Tb.

3.5.1.6 autres

Finalement NTFS offre d'autres services qui ne seront pas étudiés ici : gestion de quotas, disques amovibles, active directory (contrôle des ressources réseau), stockage étendu (bandes) ...

3.5.2 Structure d'une partition NTFS

La manière dont un système de fichiers NTFS est organisé est peu documentée, c'est la raison pour laquelle nous allons fournir une description simplifiée de ce système.

Comme pour FAT, l'unité d'allocation est le Cluster (512 bytes à 64 Kb). La taille de 4 Kb est assumée comme un standard de fait.

3.5.2.1 LCN et VCN

Les clusters d'une partition sont numérotés à partir de 0, ce nombre est appelé LCN (Logical Cluster Number). Le LCN est le numéro d'ordre du cluster dans la partition. Par ailleurs, un fichier est composé d'une suite de clusters non nécessairement contigus et chaque cluster au sein du fichier a également un numéro d'ordre qui débute à 0 ce numéro est appelé VCN (Virtual Cluster Number). Le VCN correspond à la position du cluster dans le fichier. Il n'y a que un LCN 0 sur la partition et autant de VCN 0 qu'il y a de fichiers.

LCN :

un LCN (Logical Cluster Number) est le numéro d'ordre d'un cluster dans la partition. Si une partition a 4096 clusters, ceux-ci auront des LCN allant de 0 à 4095, à partir du LCN, en connaissant la taille des clusters T, on peut calculer l'adresse disque :

adresse du cluster = LCN * T

VCN :

un VCN (Virtual Cluster Number) sert à numéroter les clusters au sein d'un fichier : le premier cluster d'un fichier est le VCN 0 quel que soit sa position sur le disque et donc son LCN.

Alors qu'il y a un lien entre le LCN et l'adresse d'un cluster, il n'y a aucun lien direct entre VCN et adresse réelle du cluster. Ce lien est établi grâce à l'attribut $DATA_RUN qui fait partie de la description du fichier, nous détaillerons cela plus loin.

3.5.2.2 Zones

Une partition NTFS contient quatre (55) zones :

- le fichier de Boot ($Boot) qui démarre toujours au secteur 0 (c'est le seul dont l'adresse est figée), en plus des informations de taille on y trouve notamment une référence vers la MFT et la MFTMirr ainsi que la taille des enregistrements de la MFT et des index donnée en nombre de clusters.
- l'espace « réservé » à la MFT (Master File Table : fichier $MFT contenant la table de tous les fichiers). Cet espace permet de garder tant que possible une MFT d'un seul tenant (56) (non fragmentée).
- Le cluster du Milieu de la partition, réservé au fichier $MFTmirr : copie des 4 premiers enregistrements de la MFT.
- Le restant du disque est réservé aux fichiers système et autres.

Lorsqu'un volume est formaté avec NTFS, plusieurs fichiers système sont crées dans le répertoire racine du volume NTFS. Ces fichiers système peuvent être stockés à n'importe quel emplacement physique du volume NTFS à l'exception du $Boot qui lui démarre toujours au secteur 0.

La MFT est elle même un fichier, elle contient la description de tous les fichiers de la partition y compris d'elle même. La MFT est le fichier de méta données le plus important.

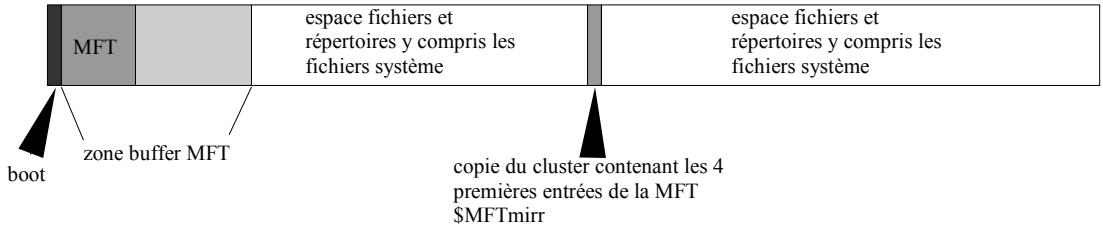


fig 3.5.2.2.1 : structure de la MFT

En faisant tourner l'utilitaire de défragmentation du disque sur une partition NTFS, les zones allouées aux fichiers système dont la MFT en début et en milieu du disque sont identifiables

3.5.3 Structure de la MFT

La MFT est subdivisée en enregistrements de taille fixe (1 à 4K) et chaque fichier y est décrit par un ou plusieurs de ces enregistrements. Toutes les informations concernant un fichier y sont décrites. Les 16 premiers enregistrements décrivent des fichiers d'organisation interne : ce sont des méta données sur le système de fichiers, leur nom commence par $, le premier d'entre eux est le descripteur de la MFT elle-même. Les 16 premières entrées de la MFT correspondent chacune à un

55 En fait il existe une cinquième zone sur le disque et c'est la copie du secteur de Boot. Celui-ci se trouve à la fin de la partition dans les versions plus récentes de NTFS. Cette zone n'est pas mentionnée dans la MFT, elle apparaît toutefois comme « utilisée » dans la table des bits (fichier $BITMAP) correspondant à l'utilisation des clusters.

56 Lorsque l'espace fichier vient à manquer, la zone MFT peut être réduite au profit de cet espace pour être récupérée éventuellement plus tard.

fichier système responsable d'un aspect de l'organisation du système de fichiers et sont les seules à avoir une position fixe dans la MFT.

Le premier enregistrement décrivant un fichier est appelé « enregistrement de Base », les suivants « extensions ».

Chaque enregistrement de la MFT est composé d'un en-tête et de plusieurs attributs, parmi les attributs d'un fichier on trouve notamment l'attribut $DATA qui contient les données du fichier.

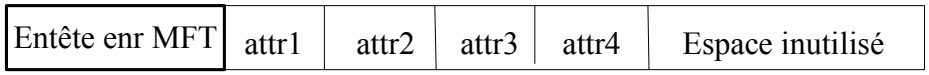


fig 3.5.3.1.1 :un enregistrement de la MFT

3.5.3.2 en-tête d'un enregistrement de la MFT

Comme les informations dans la MFT ont une taille variable, l'entête de chaque enregistrement contient une information de taille. En voici une description partielle :

- 4 bytes - taille réelle de l'enregistrement (dépend du nombre et de la taille des attributs)
- 4 bytes - taille allouée de l'enregistrement (multiple de la taille d'un enregistrement MFT)
- 2 bytes - position du premier attribut
- 2 bytes - flags
  - 0x0001 utilisé
  - 0x0002 répertoire
  - ...
-
- 2 bytes - compteur de liens hard (plusieurs noms associés au fichier)
- ...

3.5.4 Représentation de « petits fichiers » (immédiats)

Une différence fondamentale entre le système de fichiers NTFS et FAT est que dans ce dernier, toutes les informations sont stockées sous forme de fichier y compris les informations système qui se trouvent dans des fichiers de méta-données dont le nom commence par $.

NTFS représente un fichier par une entrée dans la MFT composée d'un en-tête et de 4 attributs dont le nom et les données du fichier.

3.5.4.1 attributs d'un fichier

Un fichier sur NTFS est composé des 4 attributs suivants :

Type	*Description*	*résident*
0x10	$STANDARD_INFORMATION	oui
0x30	$FILE_NAME	oui

0x50	$SECURITY_DESCRIPTOR	Abandonné en W2K
0x80	$DATA	

$STANDARD_INFORMATION : informations standard

- date-heure de création
- date-heure de modification des données
- date-heure de modification de l'entrée MFT
- date-heure de la dernière lecture (demande une **écriture disque à chaque lecture !** La fonctionnalité peut être inhibée)
- id du propriétaire
- id sécurité (il s'agit d'une référence au descripteur de sécurité dans le fichier de sécurité, permet d'associer des **droits d'accès** au fichier)
- poids de quota (0 ou taille du flux de données)
- permissions
  - 0x0001 – lecture
  - 0x0002 – caché
  - 0x0004 – système
  - 0x0020 – archive
  - 0x0040 – matériel (« device »)
  - 0x0080 – normal
  - 0x0100 – temporaire
  - 0x0200 – creux
  - 0x0400 – reparse point (sorte de lien)
  - 0x0800 – comprimé
  - 0x1000 – offline
  - 0x2000 – contenu non indexé
  - 0x4000 - crypté

$FILE_NAME : le nom et une série d'informations liées à celui-ci. Un fichier peut avoir plusieurs noms dans le cas des noms MS-DOS ou des liens hardware.

On lie au nom des informations comme

- sa longueur
- la référence à l'entrée MFT du répertoire parent (celui juste au dessus) en effet il peut être différent d'un nom à l'autre dans le cas de **liens hardware**
- de nouveaux flags (répertoire/fichier, comprimé/non comprimé, caché..)

- le nom à proprement parler

$SECURITY_DESCRIPTOR : informations de sécurité

- propriétaire du fichier
- permissions accordées par le propriétaire aux autres utilisateurs
- description des actions qui nécessitent d'être inscrites dans un « fichier journal » (logfile)

Actuellement cet attribut n'est plus dans la MFT, mais dans le fichier $Secure, on retrouve sa description grâce à l' id sécurité ($STANDARD_INFORMATION).

$DATA : les données.

NTFS permet d'avoir plus d'un attribut $DATA par fichier (57).

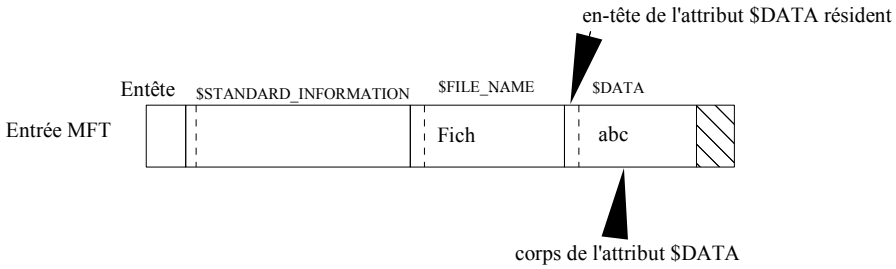


fig 3.5.4.1.1 : enregistrement MFT pour un fichier immédiat (attribut $DATA résident)

Ici, les données du fichier sont contenues dans l'entrée de la MFT qui le décrit. On pourrait croire que en NTFS, il n'existe qu'un seul fichier qui contient tous les autres : la MFT, mais il n'en est rien.

Cette représentation de l'information convient pour des fichiers de petite taille, plus précisément lorsque leur taille permet de ne pas déborder de l'enregistrement de la MFT (souvent 1K). Lorsque les fichiers ont une plus grande taille, les blocs de données sont stockés ailleurs que dans la MFT. Dans la MFT on a alors une description de leur emplacement. On dit, dans ce cas, dans la terminologie NTFS, que l'attribut $DATA est « non résident ». Ce cas de figure est décrit plus loin.

3.5.5 Codage des attributs

NTFS utilise des attributs pour décrire les fichiers dans la MFT. L'intérêt réside essentiellement dans la souplesse de cette représentation. En effet un attribut est une description dont la taille est non figée (les informations stockées dans les entrées de la MFT n'ont pas toutes la même taille et même un attribut qui décrit le même type d'information peut avoir plusieurs tailles différentes (nom de fichier). De plus tout est mis en œuvre pour que l'on puisse élargir l'ensemble des attributs connus, ce qui ouvre la porte à des évolutions du système de fichiers.

Un attribut est décrit en deux parties :

- l'en-tête
- le corps

57 Généralement un répertoire n'a pas d'attribut $DATA alors que c'est un attribut obligatoire pour un fichier, même si il peut être vide.

L'en-tête d'un attribut contient notamment :

- l'identifiant de l'attribut
- la longueur (variable)
- un flag résident/non résident (l'attribut est résident ou non)
- datarun éventuel(description d'un l'attribut non résident)
- crypté, comprimé, creux (uniquement pour un attribut $DATA - données d'un fichier non résident)
- ...

3.5.5.1 attribut résident

Un attribut est résident si sa description est entièrement contenue dans l'entrée de la MFT.

3.5.5.2 attribut non résident

Les attributs non résidents ont leur entête dans la MFT et leur corps dans une ou plusieurs zones de clusters appelées « runs », ailleurs sur le disque. La description de l'emplacement des « runs » est donnée dans le champ $DATA_RUN situé dans l'entête de l'attribut.

Le champ $DATA_RUN, permet donc de trouver la localisation sur disque du corps de l'attribut, ce champ a un rôle très général qui s'applique à n'importe quel attribut pouvant être non résident.

Quand ce champ est présent dans l'attribut $DATA d'un fichier il joue le rôle de table d'index pour les « runs » qui contiennent les données du fichier, un peu comme le faisait la FAT pour les clusters.

Chaque « run » est décrit par 2 informations : localisation et longueur du run. Chaque « run » est composé d'un nombre de clusters différent.

La représentation par « runs » est utilisée pour la description des données de grands fichiers.

Exemple de data run

Dans l'exemple suivant la localisation des « run » est donnée sous forme de décalage par rapport au début du « run » précédent plutôt que par son LCN, pour cette raison on parle plutôt de décalage. Par exemple si le premier run commence en LCN 100 et le deuxième en LCN 147 leurs début/décalage respectifs seront 100 et 47.

De plus, les champs « décalage » et « longueur » ont eux mêmes une longueur variable. Un byte (2 hexadécimaux) est réservé pour spécifier leur longueur respective.

Le descripteur de chaque run est composé de 4 informations (longueur du champ « longueur » (1 chiffre héxadécimal - 4 bits), longueur du champ « décalage » (1 chiffre héxadécimal - 4 bits), champ «décalage», champ « longueur »).

Exemple de description de run :

12 18 56 34

- en-tête sur un byte (longueur fixe) = 0x12 – deux héxadécimaux, le champ longueur codé ici sur 1 byte et , le champ décalage codé ici sur 2 bytes
- longueur : = 0x18 (1 byte)

- décalage = 0x5634 (2 bytes)

cette souplesse, permet d'adresser des fichiers de très grande taille sans perdre inutilement de la place dans les descripteurs.

3.5.5.3 la liste d'attributs ($ATTRIBUTE_LIST)

Certains attributs sont obligatoirement résidents de par leur rôle (...). Lorsque on n'arrive pas à les stocker tous dans un enregistrement de la MFT, on se sert d'un attribut nommé $ATTRIBUTE_LIST pour l'éclatement des attributs en plusieurs entrées MFT.

3.5.6 Représentation de « fichiers plus grands »

Comme nous l'avons vu plus haut, une particularité intéressante de l'attribut $DATA est qu'il est parfois résident. En effet, quand les données d'un fichier sont peu volumineuses au point de pouvoir être décrites directement dans l'entrée MFT, on préfère les stocker directement dans l'entrée MFT, ce qui a comme conséquence que l'accès aux données est plus rapide (pas de déplacement de têtes de lectures).

Dans le cas où les données sont trop volumineuses pour être stockées en résident, on utilise des « data run » (sorte de table d'index) pour retrouver les clusters où sont les données. Les « data runs » sont décrits dans le champ $DATA_RUN de l'en-tête de l'attribut $DATA quand celui-ci est non résident.

3.5.6.1 exemples de fichiers non résidents

L'attribut $DATA_RUN décrit l'entièreté des données d'un fichier qui peut se trouver sur plusieurs zones de clusters. L'attribut décrit donc plusieurs séquences de 4 valeurs terminées par un byte nul qui indique la fin des « runs » et du fichier. Nous reportons ici 3 des 4 premiers exemples (58) repris dans http://data.linux-ntfs.org/ntfsdoc.pdf

Exemple 1 – Un fichier non fragmenté

data runs:

- 12 18 56 34 00
- 12 18 56 34 - 00 (information regroupée par blocs de clusters)

run 1:

- en-tête sur un byte (longueur fixe) = 0x12 - 1 byte longueur, 2 byte décalage
- longueur = 0x18 (1 byte)
- décalage = 0x5634 (2 bytes)

fin:

- en-tête = 0x00 – fin des données

résumé:

- 0x18 Clusters @ LCN 0x5634

Il s'agit donc ici d'un fichier non fragmenté, de taille 0x18 clusters, qui débute en LCN 0x5634.

58 Nous avons simplifié l'exemple donné sur le site en omettant la représentation des entiers en « little endian ».

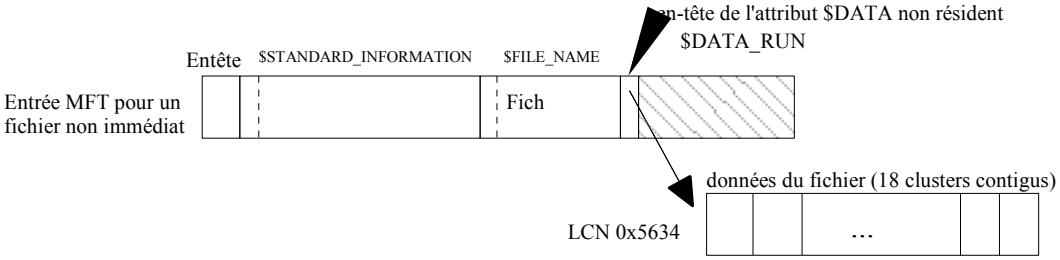


fig 3.5.6.1.1 :enregistrement MFT pour un fichier grand fichier non fragmenté

Exemple 2 – Un fichier fragmenté

data runs:

- 13 38 34 25 73 23 01 14 02 11 E5 13 42 03 00 AA 00
- 13 38 34 25 73 - 23 01 14 02 11 E5 - 13 42 03 00 AA – 00 (information regroupée par blocs de clusters)

run 1:

- en-tête = 0x13 - 1 byte longueur, 3 byte décalage
- longueur = 0x38 (1 byte)
- décalage = 0x342573 (3 bytes)

run 2:

- en-tête = 0x23 - 2 byte longueur, 3 byte décalage
- longueur = 0x114 (1 byte)
- décalage = 0x363758 (0x211E5 relativement à 0x342573)

run 3:

- en-tête = 0x13 - 1 byte longueur, 3 byte décalage
- longueur = 0x42 (1 byte)
- décalage = 0x393802 (0x300AA relativement à 0x363758)

fin:

- en-tête = 0x00 – fin des données

résumé:

- 0x38 Clusters @ LCN 0x342573
- 0x114 Clusters @ LCN 0x363758
- 0x42 Clusters @ LCN 0x393802

Il s'agit donc ici d'un fichier fragmenté, de taille 0x18E clusters, avec des blocs de données qui débutent en LCNs 0x342573, 0x363758 et 0x393802.

Exemple 3 – Un fichier creux

Un fichier creux est un fichier qui contient des espaces sans données (à ne pas confondre avec des espaces blancs d'un fichier de texte qui contiennent des blancs). Des espaces sans données sont des espaces où l'on a jamais rien écrit. Un programme qui écrit dans un fichier peut demander de se positionner plus loin dans le fichier avant de continuer à écrire, il crée ainsi un trou d'information, un traitement de texte, ne permet pas de créer de tels fichiers.

data runs:

- 11 30 20 10 60 11 10 30 00
- 11 30 20 - 10 60 - 11 10 30 - 00 (information regroupée par blocs de clusters)

run 1:

- en-tête = 0x11 - 1 byte longueur, 1 byte décalage
- longueur = 0x30 (1 byte)
- décalage = 0x20 (1 bytes)

run 2:

- en-tête = 0x10 - 1 byte longueur, 0 byte décalage (trou)
- longueur = 0x60 (1 byte)
- décalage = -/-

run 3:

- en-tête = 0x11 - 1 byte longueur, 1 byte décalage
- longueur = 0x10 (1 byte)
- décalage = 0x50 (0x30 relativement à 0x20)

fin:

- en-tête = 0x00 – fin des données

résumé:

- 0x30 Clusters @ LCN 0x20
- 0x60 Clusters trou
- 0x10 Clusters @ LCN 0x50

Il s'agit donc ici d'un fichier creux non fragmenté, de taille 0xA0 clusters, avec des blocs de données qui débutent en LCNs 0x20 et 0x50.

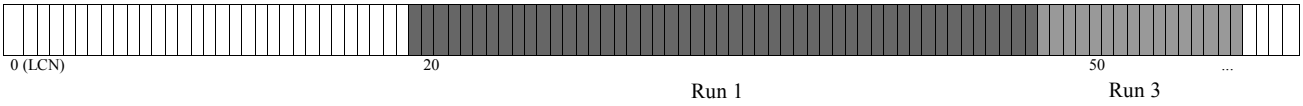


fig 3.5.6.1.2 : vision de l'espace disque : utilisation réelle des clusters (LCN) pour un fichier creux

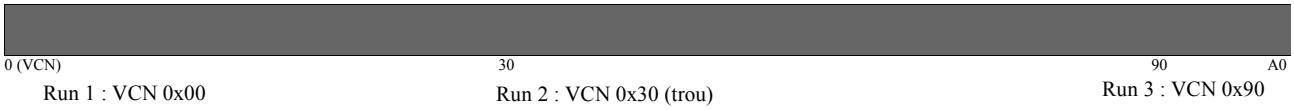


fig 3.5.6.1.3 : vision des clusters virtuels (VCN) d'un fichier creux

Il est intéressant de constater que le trou (run 2) n'utilise pas de clusters sur le disque. NTFS n'utilise pas d'espace disque pour représenter les trous des fichiers creux.

3.5.6.2 liens hard

NTFS permet d'associer plusieurs noms à un fichier : c'est le cas lorsque le système attribue à un fichier un nom raccourci compatible avec les noms de MS-DOS.
NTFS implémente de cette manière les liens hardware.

Quand on supprime un fichier lien hardware, son nom est supprimé de la MFT, ce n'est que lors de la suppression du dernier nom (59) que le fichier sera réellement supprimé.

3.5.6.3 liens soft

NTFS permet également les « liens software » au travers des « reparse points » : un mécanisme d'indirection.

3.5.6.4 liste des blocs libres ($Bitmap)

Lorsque NTFS a besoin d'allouer des nouveaux blocs à un fichier il doit pouvoir différencier les

59 un compteur de liens hardware est prévu dans l'entête de l'enregistrement de la MFT.

blocs libres des blocs alloués. Il n'existe pas réellement de liste de blocs libres, mais une table de bits avec l'état d'occupation des clusters sur le disque (0 : libre, 1 occupé). La table est le fichier système $Bitmap. Le système l'utilise pour allouer du nouvel espace à un fichier et pour libérer de l'espace.

Il est intéressant de noter que les clusters appartenant à la zone MFT sont particuliers, dans le sens où ils ne seront pas alloués normalement à un fichier. L'allocation à un fichier d'un de ces clusters ne peut arriver que si le restant du disque est plein. Dans ce cas la zone MFT est réduite de moitié.

3.5.6.5 liste des blocs endommagés ($BadClus)

Cette liste est mémorisée dans le fichier système $BadClus. Celui-ci est mis à jour à chaque fois qu'on rencontre un cluster défectueux (avec un secteur défectueux). Ce fichier est un « fichier creux » qui a comme taille Virtuelle la taille de la partition. Les trous correspondent aux clusters en bon état et les données aux clusters défectueux. De plus un cluster défectueux est également marqué comme utilisé dans le fichier $Bitmap pour empêcher toute tentative nouvelle allocation. Si lors d'une lecture ou une écriture le système détecte un cluster défectueux, un nouveau cluster est alloué automatiquement au fichier. Dans le cas d'une lecture, les données du cluster seront probablement perdues, une erreur est signalée au programme.

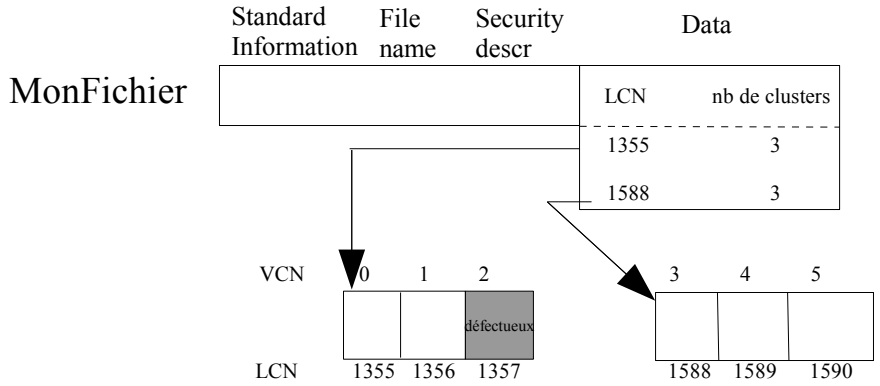


fig 3.5.6.5.1 : fichier avec un cluster défectueux

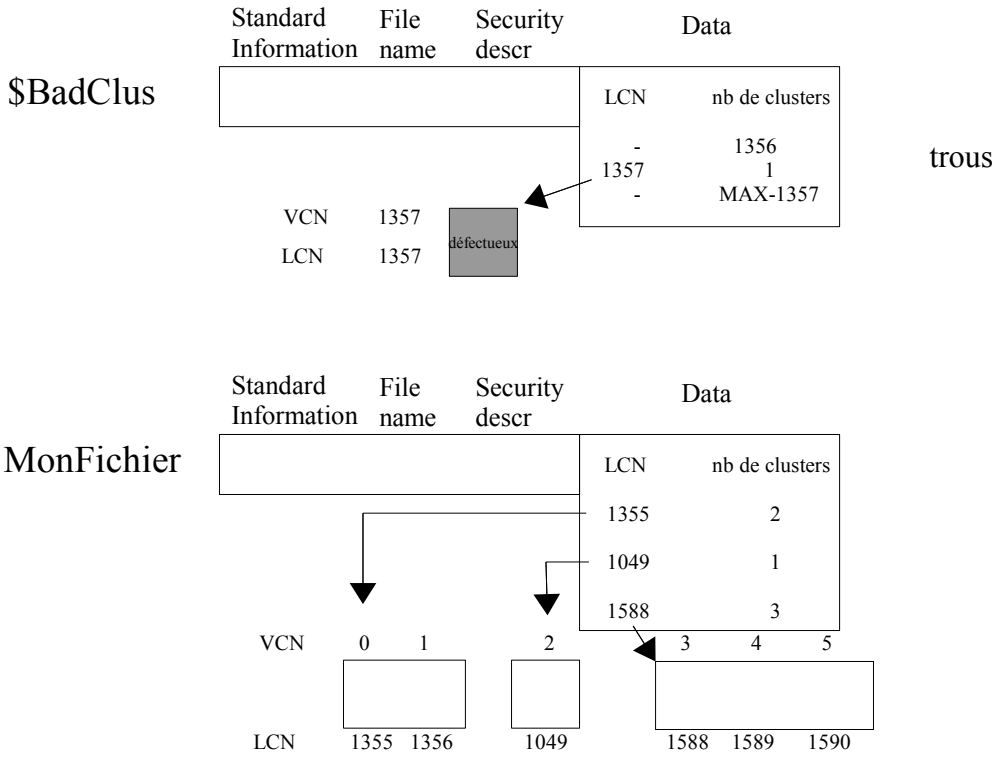


fig 3.5.6.5.2 : Après isolement et allocation d'un nouveau cluster au fichier

3.5.7 Représentation de répertoires

3.5.7.1 structure

Un répertoire sous NTFS a une entrée dans la MFT car c'est également un fichier. Son attribut le plus important est un « index » basé sur les noms de fichiers (60). Ces « index » permettent l'obtention rapide de la liste de fichiers triée (sur le nom) et en très peu de lectures (donc peu de temps) l'obtention du descripteur d'un fichier du répertoire, même si ce répertoire contient des milliers de fichiers.

Les attributs d'un répertoire :

Type	*Description*	*Nom*
0x10	$STANDARD_INFORMATION	
0x30	$FILE_NAME	Nom du répertoire
0x50	$SECURITY_DESCRIPTOR	obsolète sur W2K
0x90	$INDEX_ROOT	$I30
0xA0	$INDEX_ALLOCATION	$I30
0xB0	$BITMAP	$I30

60 L'index est implémenté sous forme d'un arbre balancé B+ (B+ tree)

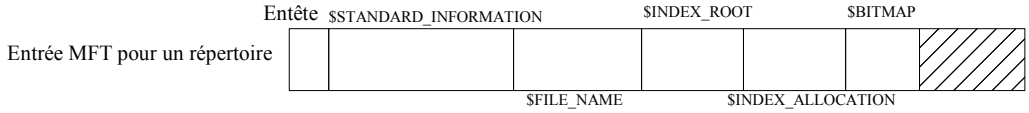


fig 3.5.7.1.1 : enregistrement MFT pour un répertoire

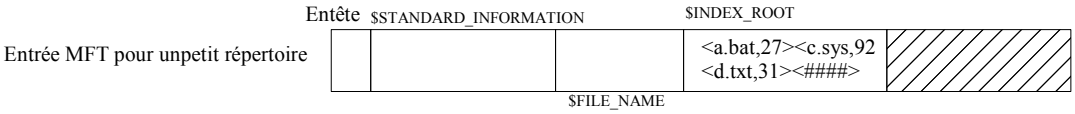


fig 3.5.7.1.2 : un petit répertoire de trois fichiers

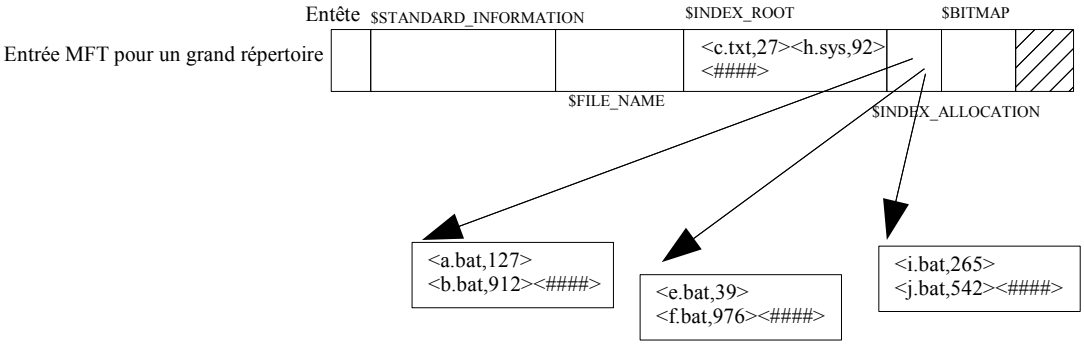


fig 3.5.7.1.3 : un grand répertoire

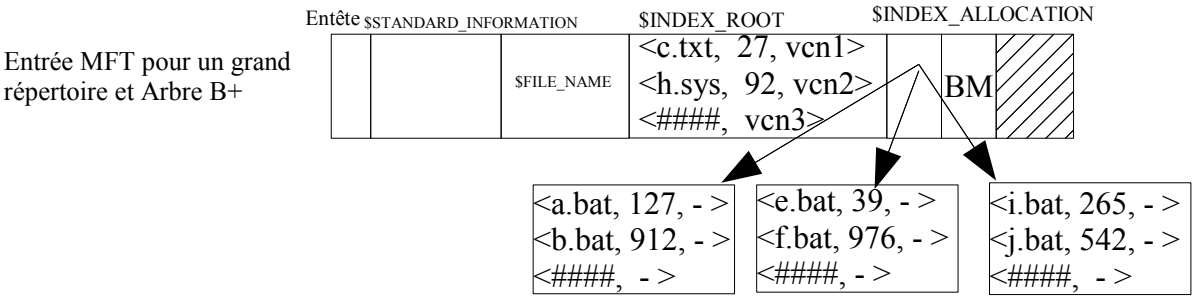


fig 3.5.7.1.4 : un grand répertoire et ses B+ index

3.5.7.2 répertoire racine (« $. »)

Tout comme pour le système de fichiers FAT, le système NTFS a besoin d'un point de départ pour retrouver ses fichiers et ce point de départ est le répertoire racine. NTFS utilise également une convention pour ce répertoire : la description du répertoire « $. » se trouve en sixième position dans la MFT.

3.5.8 Liste des attributs

La liste que voici, est la liste des Attributs qui se trouve dans le fichier $AttrDef ce sont les attributs

connus par le système NTFS

Type	*OS*	*nom*
0x10		$STANDARD_INFORMATION
0x20		$ATTRIBUTE_LIST
0x30		$FILE_NAME
0x40	NT	$VOLUME_VERSION
0x40	2K	$OBJECT_ID
0x50		$SECURITY_DESCRIPTOR
0x60		$VOLUME_NAME
0x70		$VOLUME_INFORMATION
0x80		$DATA
0x90		$INDEX_ROOT
0xA0		$INDEX_ALLOCATION
0xB0		$BITMAP
0xC0	NT	$SYMBOLIC_LINK
0xC0	2K	$REPARSE_POINT
0xD0		$EA_INFORMATION
0xE0		$EA
0xF0	NT	$PROPERTY_SET
0x100	2K	$LOGGED_UTILITY_STREAM

3.5.9 Fichiers Système (méta données)

Entrée MFT	*nom*	*OS*	*Description*
0	$MFT		Master File Table – Table d'index de tous les fichiers
1	$MFTMirr		Copie de 4 premières entrées de la MFT
2	$LogFile		Logging file transactionnel
3	$Volume		Numéro de série, date de création,...
4	$AttrDef		Définition des attributs connus
5	. (dot)		Répertoire racine
6	$Bitmap		Table d'allocation des clusters
7	$Boot		Boot record du volume (partition bootable)
8	$BadClus		Liste des clusters endommagés
9	$Quota	NT	Information de Quota
9	$Secure	2K	Descripteurs de sécurité utilisés pour le volume
10	$UpCase		Correspondance majuscules pour l'Unicode
11	$Extend	2K	Un répertoire: $ObjId, $Quota, $Reparse, $UsnJrnl
12-15	<non utilisé>		Non utilisé vide
16-23	<non utilisé>		Non utilisé
quelconque	$ObjId	2K	Id unique du fichier
quelconque	$Quota	2K	Information sur les quotas (par personne et volume)
quelconque	$Reparse	2K	Reparse point information

quelconque	$UsnJrnl	2K	Journalling of Encryption
> 24	A_File		fichier
> 24	A_Dir		répertoire
...	...		...

3.5.10 Fiabilité

3.5.10.1 résistance aux pannes

Si on compare au système FAT, où la zone critique en cas de secteurs endommagés va du premier secteur du disque jusque à la fin de la FAT, ici la zone critique est réduite au au secteur de boot, en effet la MFT n'a pas un emplacement fixe à l'intérieur de la zone qui lui est réservée. Pour un système FAT32, d'une partition de 40Gb avec des clusters de 4K, la zone critique en début de disque s'étendrait à 320 Mb de taille.

3.5.10.2 journalisation et récupération de données

NTFS utilise la technique du « logging » pour assurer la cohérence des structures sur le disque. Chaque opération modifiant la structure du système de fichiers peut être subdivisée en un ensemble de sous-opérations. Un ensemble de sous-opérations formant une seule opération logique est aussi appelé « transaction » (61).

Le principe d'une transaction est que celle-ci doit être réalisée entièrement ou pas du tout. Ainsi, après une panne, le système doit pouvoir retrouver toutes les transactions qui sont partiellement réalisées et choisir entre les défaire complètement (rollback) ou les terminer.

En NTFS, toutes les sous-opérations sont écrites dans un fichier spécial : $LogFile avant d'être écrites à leur places définitives. Une fois la transaction complétée, c'est également marqué dans le fichier de log. Le système, interrompu brutalement, pourra détecter les transactions inachevées et retrouver un état cohérent en défaisant ou en refaisant certaines opérations. Même si NTFS possède toujours l'utilitaire chkdsk, son utilisation est souvent redondante avec le système journalisé.

3.5.11 Sécurité

NTFS utilise des ACL (Access Control List) pour sécuriser ses fichiers. Une liste de descripteurs de sécurité est associée à chaque fichier.

La solution adoptée dans les premières versions de NTFS consistait à mémoriser un attribut $SECURITY_DESCRIPTOR pour chaque fichier et répertoire contenant notamment les informations:

- propriétaire
- permissions accordées aux autres utilisateurs
- actions sujettes à être mémorisées dans un fichier journal (logfile) (écritures , accès...)
- ...

61 NTFS emprunte le concept de transaction aux bases de données relationnelles.

Comme la plupart du temps, les administrateurs confèrent les mêmes droits d'accès à tous les fichiers d'un même répertoire l'information de sécurité ainsi stockée est répétée un grand nombre de fois, ce qui résulte en une grosse perte de place sur le disque.

Dans les versions ultérieures de NTFS on a amélioré l'utilisation d'espace disque en centralisant toutes les valeurs possibles de descripteur de sécurité dans un fichier nommé $Secure. Il n'y a donc plus qu'une seule instance de descripteur de sécurité, même si celui-ci est partagé par tous les fichiers d'un répertoire. Dans la description des fichiers on a ajouté une référence vers le fichier $Secure : le « Security Id ».

Le fichier $Secure

Le fichier $Secure contient deux index : les attributs $SDH et $SDI et un flux de données : l'attribut $SDS.

$SDH est un index qui permet de retrouver (par une technique de hashing) un descripteur dans la partie des données du fichier ($SDS).

$SDI est un index basé sur in identifiant du descripteur : le « security Id » (à ne pas confondre avec le SID).

Cette configuration, nous permet d'accéder des deux manières différentes au fichier de sécurité, ces deux manières correspondent aussi à deux utilisations différentes du système de fichiers :

- retrouver un descripteur de sécurité existant pour éventuellement le partager (lors de la création du fichier ou lors de la modification de ses droits)
- valider les droits d'accès à un fichier ou répertoire au moment de son utilisation par un processus (lecture, écriture,...)

Dans le premier cas on se servira en premier lieu du SDH qui permet de retrouver le descripteur de sécurité dans le fichier $Secure et de l'y ajouter le cas échéant. Dans le cas où le descripteur est ajouté au fichier, les deux index seront mis à jour également.

De plus, dans ce cas, le « security Id » est noté dans l'attribut $STANDARD_INFORMATION de l'entrée MFT du fichier. Le security Id est codé sur 32 bits, ce qui est de loin inférieur à l'espace utilisé par le $SECURITY_DESCRIPTOR, d'où le gain de place sur le disque.

Il est à remarquer que un descripteur de sécurité qui n'est plus utilisé n'est pas supprimé du fichier $Secure.

Dans le deuxième cas (le plus fréquent) on sera amené à rechercher le descripteur de sécurité à partir du « security id » au moyen du deuxième index : le $SII.

Une fois le descripteur trouvé NTFS pourra vérifier la validité de l'accès demandé en comparant les permission et l'identifiant de sécurité du processus demandeur (user, group..) le SID (Security Id Descriptor).

Attributs du fichier $Secure :

Type	*Description*	*Name*
0x10	$STANDARD_INFORMATION	
0x30	$FILE_NAME	$Secure
0x80	$DATA	$SDS

0x90	$INDEX_ROOT	$SDH
0x90	$INDEX_ROOT	$SII
0xA0	$INDEX_ALLOCATION	$SDH
0xA0	$INDEX_ALLOCATION	$SII
0xB0	$BITMAP	$SDH
0xB0	$BITMAP	$SII

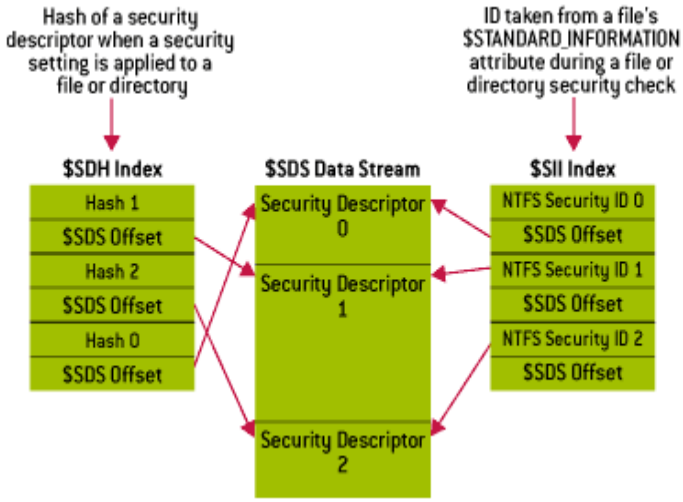


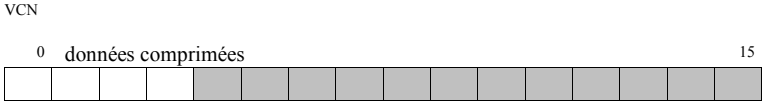
fig 3.5.11.1.1 : accès aux descripteurs de sécurité du fichier $Secure

3.5.12 Compression de données

Les fichiers sur NTFS peuvent être s. Un flag apparaît à deux endroits : au niveau de l'attribut $STANDARD_INFORMATION et dans l'entête de l'attribut $DATA.

Nous avons déjà vu un premier cas de compression pour les fichiers creux, mais NTFS permet également de comprimer des fichiers de données non creux.

Un fichier comprimé est divisé en blocs de 16 clusters décrits séparément. Sur chacun de ces blocs de données on applique l'algorithme de compression, qui est plus ou moins efficace. Si on parvient à réduire le nombre de clusters de un de ces blocs, on écrira sur disque les données comprimées, sinon on écrira les données en clair. Ces blocs de clusters sont décrits dans un $DATA_RUN, dont chaque entrée est associée à un bloc de taille de départ 16. Certains « runs » auront une taille effective inférieure à 16 clusters grâce à la compression de données. Les écritures et lectures de données sur le disque, se feront par blocs de 16 clusters et à ce moment les algorithmes de compression/décompression seront appliqués.



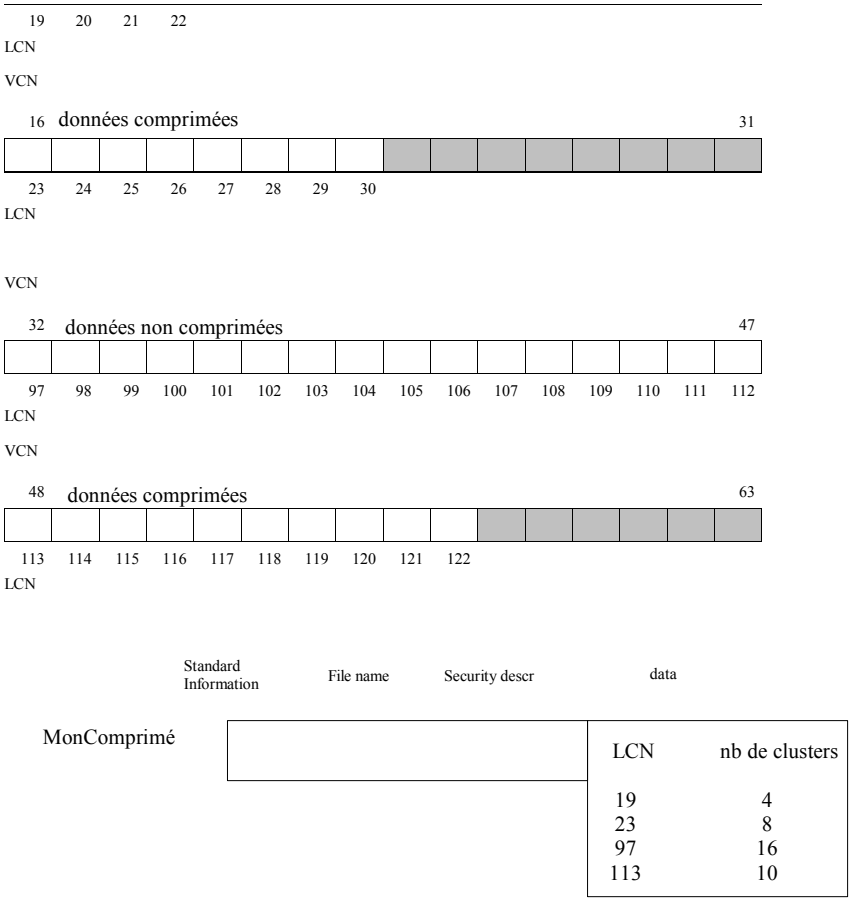


fig 3.5.12.1.1 : un fichier comprimé et son « data run »

De plus, par l'utilisation d'une mémoire cache, l'écriture sur disque d'un fichier comprimé prendra le même temps qu'un autre fichier, en effet l'écriture sur disque et la compression sont faits de manière asynchrone.

3.5.13 Encryptage

NTFS permet l'encryptage de données, un flag de l'entête de l'attribut $DATA signale si le flux de données est crypté ou non.

3.5.14 Quotas

NTFS permet de limiter la place occupée sur le disque par utilisateur. A chaque fichier est associé un champ contenant le poids en nombre de bytes que le fichier a pour cette comptabilisation. Ce champ est mémorisé dans l'attribut $STANDARD_INFORMATION.

3.6 Testez votre compréhension

1. Pourquoi ne peut-on pas organiser une partition de 60 Gb en utilisant une FAT16 ? Quelle taille de cluster devrait-on choisir ? Expliquez.
2. Quelle est la taille maximum d'un fichier en FAT ? Comment justifiez-vous cela ?
3. Décrivez le travail du système d'exploitation pour localiser le fichier a:\sys1\exemple.txt sur une FAT16;-).
4. Soient les informations suivantes données dans le secteur de boot d'un système de fichiers FAT :
   - ✔ nombre de secteurs par cluster (NSC)
   - ✔ taille zone réservée (boot) (en secteurs) (TZR)
   - ✔ nombre de FAT (NF)
   - ✔ nombre d'entrées dans le répertoire racine(NERR)
   - ✔ nombre de secteurs dans la FAT (NSF)
   - ✔ taille d'un secteur (TS)
   - ✔ nombre total de secteurs(NS)
   - ✔ ...

   1. Sachant que les clusters sont numérotés à partir de 2, donnez la formule qui permet au système de trouver le numéro de secteur correspondant au cluster N (N >= 2) ?
   2. Donnez un schéma de la structure d'un système de fichiers FAT pour construire votre réponse.
5. Expliquez de quelles informations a besoin le système pour connaître la position du premier et du deuxième cluster d'un fichier donné a:\fich1 et a:\monRep\fich2 dans une partition FAT. Où se trouve le numéro du premier cluster du fichier. Détaillez la démarche à partir du chemin complet du fichier.
6. Quelle information dans la description d'un fichier sous NTFS permet de retrouver ses données ? Expliquer. Y a-t-il une différence selon la taille du fichier ?
7. Quelle différence y a-t-il entre la représentation d'un attribut résident et non résident de NTFS ?
8. Expliquez le mécanisme utilisé par NTFS pour éviter l'utilisation de mauvais secteurs.
9. Expliquez le principe de la compression de données en NTFS.
10. Donnez votre interprétation du champ DataRun suivant :

    221000A000101012AA202B00

    1. Correct ?
    2. Nombre de RUN ?
    3. Description de chaque RUN ?
    4. Fichier fragmenté ?
    5. Fichier creux ?
    6. Fichier comprimé ?
    7. Nombre de clusters du fichier (virtuel et sur le disque) ?

4 Bibliographie

1. [TNB] - « Systèmes d'exploitation » par Andrew Tanenbaum – *cet ouvrage est régulièrement réédité. Référence incontournable (pour les cours de système d'exploitation de première, deuxième et ...)*.
2. [ZAN] - « Architecture et technologie des ordinateurs » par Paolo Zannella, Yves Ligier – DUNOD 2002 (ISBN 2-10-003801-X)

5 Quelques Liens

structure du MBR

http://fr.wikipedia.org/wiki/Master_boot_record

Unité centrale de traitement – interruptions

cours de M. Tisserant ESIL chapitres 7 et 8

http ://tisserant.ftp-developpez.com/chap7.pdf CPU

http ://tisserant.ftp-developpez.com/chap8.pdf interruptions

FAT

Hardware White Paper : spécification Microsoft d'une FAT 32 (en anglais)
http://download.microsoft.com/download/1/6/1/161ba512-40e2-4cc9-843a-923143f3456c/fatgen103.doc

http ://www.microsoft.com/whdc/system/platform/firmware/fatgen.mspx

architecture de la FAT : article
http://technet.microsoft.com/en-us/library/cc776720.aspx#w2k3tr_fat_how_nxis

tailles par défaut de clusters FAT-NTFS-FAT32
http://support.microsoft.com/kb/140365/EN-US/ (en anglais)

http://support.microsoft.com/kb/314878/fr
http://support.microsoft.com/kb/192322/fr

NTFS

Description en anglais de sourceforge

http://data.linux-ntfs.org/ntfsdoc.pdf

« Inside Win2K NTFS » , « Exploring NTFS ON Disk Structures » - articles de M. Russinovitch (en anglais)
http://www.Windowsitpro.com/Authors/AuthorID/76/76.html -

Articles sur le fonctionnement de NTFS
http://technet.microsoft.com/en-us/library/cc758691.aspx

Comment NTFS réserve de l'espace pour sa table de fichiers maîtres (MFT). Changements en Windows XP et Windows Server 2003 : la MFT est défragmentée par les otils de défragmentation ici !
http://support.microsoft.com/kb/174619